



EGE ÜNİVERSİTESİ

EÜ FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

EŞ ZAMANLI HAMLE TABANLI BİR SATRANÇ OYUN MOTORU GELİŞTİRMESİ

Mustafa TÜRK

Tez Danışmanı : Prof. Dr. Mehmet Emin DALKILIÇ

Uluslararası Bilgisayar Anabilim Dalı

Bilim Dalı Kodu : 619.02.04
Sunuş Tarihi : 22.12.2016

Bornova-İZMİR

2016

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

EŞ ZAMANLI HAMLE TABANLI BİR SATRANÇ OYUN MOTORU GELİŞTİRMESİ

Mustafa TÜRK

Tez Danışmanı : Prof. Dr. Mehmet Emin DALKILIÇ

Uluslararası Bilgisayar Anabilim Dalı

Bilim Dalı Kodu : 619.02.04

Sunuş Tarihi : 22.12.2016

Bornova-İZMİR
2016

Mustafa TÜRK tarafından yüksek lisans tezi olarak sunulan "Eşzamanlı Hamle Tabanlı Bir Satranç Oyun Motoru Geliştirmesi" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 22 Aralık 2016 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	**: Prof. Dr. Mehmet Emin Dalkılıç**	**.....**
Raportör Üye	**: Doç. Dr. Orhan DAĞDEVİREN**	**.....**
Üye	**: Doç. Dr. Mehmet S. ÜNLÜTÜRK**	**.....**

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi / Doktora Tezi olarak sunduğum "Eş Zamanlı Hamle Tabanlı Bir Satranç Oyun Motoru Geliştirmesi" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

01 / 12 / 2016

Mustafa TÜRK

ÖZET

EŞ ZAMANLI HAMLE TABANLI

BİR SATRANÇ OYUN MOTORU GELİŞTİRMESİ

TÜRK, Mustafa

Yüksek Lisans Tezi, Uluslararası Bilgisayar Anabilim Dalı

Tez Danışmanı: Prof. Dr. Mehmet Emin DALKILIÇ

Aralık 2016, 182 sayfa

Eşit güçteki oyuncular arasında yapılan klasik satranç maçlarının sonuçları analiz edildiğinde beyaz taşlarla oynayan tarafın kazanma olasılığının rakibine kıyasla %22.19 daha yüksek olduğu görülmektedir (Banks G. 2016). Bu tez çalışmasıyla, klasik satranç oyununa göre daha adil ve eş zamanlı olaylar için strateji geliştirmeye yardımcı olacağı değerlendirilen yeni bir satranç oyunu uygulaması geliştirilmiştir. Bu uygulamanın geliştirilmesinin öncelikli sebebi: Klasik satranç oyunundaki beyaz taşlarla oynayan oyuncunun açık avantajının tespit edilmesinin ardından bu adaletsizliği ortadan kaldıran eş zamanlı bir satranç oyunu için bu tezin danışmanı Prof. Dr. M. E. DALKILIÇ tarafından önerilen bir kurallar setinin tutarlılığının test edilme gereksinimidir. Bu kurallar seti kullanılarak Grafik Kullanıcı Arayüzü (GUI) tabanlı bir web uygulaması HTML, CSS, Java Script, jQuery, Node.js, express.js ve socket.io programlama dilleri/kütüphaneleri kullanılarak geliştirilmiştir. Bu uygulamada insan-insan ve insan-bilgisayar modelleri yer almaktadır.

Sonuç olarak, bu tez çalışmasında, belirlenen kurallar seti test edilmiş ve tutarlı hale getirilmiş, eş zamanlı hamle stratejilerine dayanan yeni bir zeka oyunu geliştirilmiş ve oyun için ücretsiz bir web platformu hizmete sunulmuştur.

Anahtar sözcükler: Eş zamanlı satranç, satranç motoru, web uygulaması geliştirme

ABSTRACT

DEVELOPING A SIMULTANEOUSLY MOVE BASED CHESS APPLICATION

TÜRK, Mustafa

MSc in Information Technology

Supervisor: Prof. Dr. Mehmet Emin DALKILIÇ

December 2016, 182 pages

It is observed through the analysis of the results of the classical chess games played by players of the equivalent strength, that, players with the white pieces has a 22.19% more winning rate as compared to the players with black pieces (Banks G. 2016). In this thesis, a new chess game which is more fair than the classical chess and has been evaluated to be useful to develop strategies for simultaneous events, has been implemented. The primary reason for developing this new chess application is the necessity for testing the stability of the rule set proposed by the thesis advisor, Prof. Dr. M. E. DALKILIC who came up with a rule set for a more fair chess after seeing the unfair advantage of the first movers in classical chess. A Graphical User Interface (GUI) based web application implementing this rules set has been developed using HTML, CSS Java Script, Jquery, Node.js, express.js and socket.io programming tools and libraries. This application contains both human - human and human – computer models.

As a result, in this thesis work, proposed rule set has been tested and slightly modified for stability. Furthermore, a new intelligence game based on simultaneous moves has been developed and presented as a free web application.

Keywords: simultaneous move chess, chess engine, web application development

TEŞEKKÜR

Bu çalışma öncesinde araştırma ve geliştirme becerilerime çok önemli katkılarda bulunan, ders aldığım değerli hocalarıma ve özellikle bu çalışmanın fikrini ve temel yapı taşlarını oluşturan, aynı zamanda çalışma süresince değerli görüş ve önerilerini esirgemeyen, yol gösteren ve fikir veren değerli hocam Prof. Dr. Mehmet Emin Dalkılıç'a, teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

ÖZET ..... **vii**

ABSTRACT ..... **ix**

TEŞEKKÜR ..... **xi**

ŞEKİLLER DİZİNİ ..... **xvi**

ŞEKİLLER DİZİNİ(devam) ..... **xvii**

ŞEKİLLER DİZİNİ(devam) ..... **xviii**

ÇİZELGELER DİZİNİ ..... **xix**

1 GİRİŞ **1**

1.1 Eş Zamanlı Satranç Oyunu Nedir? 2

1.2 Tez Çalışmasının Amacı 3

1.3 Tez Çalışmasının Önemi 3

1.4 Eş Zamanlı Satranç Oyununun Kuralları 4

1.5 Eş Zamanlı Satranç Oyununun Tarihçesi 13

2 KLASİK SATRANÇ PROGRAMLAMA VE EŞ ZAMANLI SATRANÇ PROGRAMLAMA **14**

2.1 Satranç Programlamaya Genel Bakış 15

2.2 Satranç Oyun Tahtasının Bilgisayarda Temsili 15

2.3 Takip Edilmesi Gereken Bilgiler 18

2.4 Bazı Önemli Değişkenler ve Sabitler 19

3 EŞ ZAMANLI SATRANÇ OYUN MOTORU KODLAMA **21**

3.1 Sunucu ile İlgili Kodlama 21

3.2 İstemci Tarafında Uygulamanın Başlatılması 23

3.3 Oyun Tahtasının Tanımlanması 25

3.4 Satır (Ranks) ve Sütun (Files) Dizileri 30

3.5 Oyun Taşı Listeleri (Piece lists) 32

3.6 Pozisyon Özetleme Anahtarı (Position Hash Key) 33
3.7 64 ve 120 Eleman Uzunluğundaki Dizi Dönüşümleri 36
3.8 FEN Pozisyon Metni (FEN Position String) 38
3.9 Materyal Listesinin Güncellenmesi 45
3.10 Oyun Tahtasının Konsola Yazdırılması 47
3.11 Oyun Taşı Listesinin Konsola Alfa-Nümerik Olarak Yazdırılması 49
3.12 Oyun Karesinin Saldırı Altında Olup Olmadığının Kontrolü 50
3.13 Hamle Değişkeninin Yapısı 55
3.14 Hamle Listesinin Oluşturulması 59
3.15 Pozisyon Anahtarı Oluşturma Fonksiyonları 72
3.16 Oyun Taşının Tahtadan Çıkarılması 74
3.17 Oyun Taşının Tahtaya Eklenmesi 75
3.18 Bir Taşın Hareket Ettirilmesi 76
3.19 Oyun Geçmişi 77
3.20 Klasik Satrançta Hamle Yapma Fonksiyonu 78
3.21 Klasik Satrançta Hamleyi Geri Alma Fonksiyonu 83
3.22 Eş Zamanlı Hamle Yapma Fonksiyonu 85
3.23 Eş Zamanlı Satrançta Hamleyi Geri Alma Fonksiyonu 100
3.24 Yapılan Eş Zamanlı Hamle Sonucunda Oluşan Pozisyonun İncelenmesi . 106
3.25 Oyun Süresinin Tutulması (Sayaç) 110

4 GRAFİK KULLANICI ARA YÜZÜ (GUI - GRAPHICAL USER INTERFACE) 115

4.1 Grafik Ara Yüzdeki Statik Alanlar 115
4.2 Grafik Ara Yüzün Dinamik Alanları ve Yeni Oyun Kurulumu 126
4.3 FEN Metninin Girdi Alanı 151
4.4 Grafik Ara Yüzlü Oyun Tahtasının Oluşturulması 153
4.5 Grafik Kullanıcı Ara Yüzünde Hamlelerin Yapılması 160

5 EŞ ZAMANLI SATRANÇ UYGULAMASININ KULLANIMI 176

6 SONUÇLAR 181

7 KAYNAKLAR DİZİNİ 183

8 ÖZGEÇMİŞ 184

ŞEKİLLER DİZİNİ

Şekil Sayfa

Şekil 1-1 www.computerchess.org.uk/ccrl/404/ sitesi test özeti.....1

Şekil 1-2 Zamanlı Hamle Kuralı Örnek I.....4

Şekil 1-3 Eş Zamanlı Hamle Kuralı Örnek II 5

Şekil 1-4 Eş Zamanlı Şah Çekme Durumu Öncesindeki Tahta Pozisyonu....6

Şekil 1-5 Eş Zamanlı Şah Çekme Durumu 6

Şekil 1-6 Eş Zamanlı Savunma Durumu.....7

Şekil 1-7 Eş Zamanlı Mat Öncesi Tahta Durumu 8

Şekil 1-8 Eş Zamanlı Mat Durumu 8

Şekil 1-9 Şahların Komşu Karelere Düşmesi.....9

Şekil 1-10 Aynı Kareye Hareket Etme.....10

Şekil 1-11 Aynı Kareye Hareket Sonucu 11

Şekil 1-12 Tutsak Taş n. Hamle 12

Şekil 1-13 Tutsak Taş n+1. Hamle.....12

Şekil 2-1 Satranç Tahtası Temsili: 64 Elemanlı Dizi 16

Şekil 2-2 Satranç Tahtası Temsili: Kare, İndis İlişkisi.....16

Şekil 2-3 Satranç Tahtası Temsili: Taşlar ve Yerleşimi.....17

Şekil 2-4 120 Elemanlı Dizi ve Hamlelerin Tespiti 18

ŞEKİLLER DİZİNİ(devam)

Şekil Sayfa

Şekil 3-1 Satır (Rank) Dizisi 31

Şekil 3-2 Sütun(File) Dizisi..... 32

Şekil 3-3 Bir oyun karesine saldırı..... 51

Şekil 3-4 Siyah piyon saldırısı 53

Şekil 3-5 Geçerken alma karesine hamle yapma 81

Şekil 3-6 CastlePerm Dizisi 95

Şekil 3-7 Hamlelerin ekrana yazdırılması..... 98

Şekil 4-1 Kullanıcı ara yüzü - karşılama ekranı..... 122

Şekil 4-2 Kullanıcı ara yüzü - rakibin beklenmesi..... 122

Şekil 4-3 Rakip bekleyen oyuncular 124

Şekil 4-4 Oyun tahtası üzerindeki alanların görünümü..... 125

Şekil 4-5 Oyun tahtasındaki yardımcı alanlar (aside) 126

Şekil 4-6 Terfi işlemi menüsü 162

Şekil 5-1 Eş Zamanlı Satranç Ana Sayfa 176

Şekil 5-2 Oyun Kurma: Tek Oyuncu Öğrenme Modu..... 177

Şekil 5-3 Oyun Kurma: Tek Oyuncu Oyun Modu..... 177

Şekil 5-4 Tek Kullanıcı: Öğrenme Modu..... 178

ŞEKİLLER DİZİNİ(devam)

Şekil Sayfa

Şekil 5-5 İki Oyuncu Seçeneği: Oyun Kurulumu 179

Şekil 5-6 İki Oyuncu Seçeneği Oyun Ekranı 180

ÇİZELGELER DİZİNİ

Çizelge Sayfa

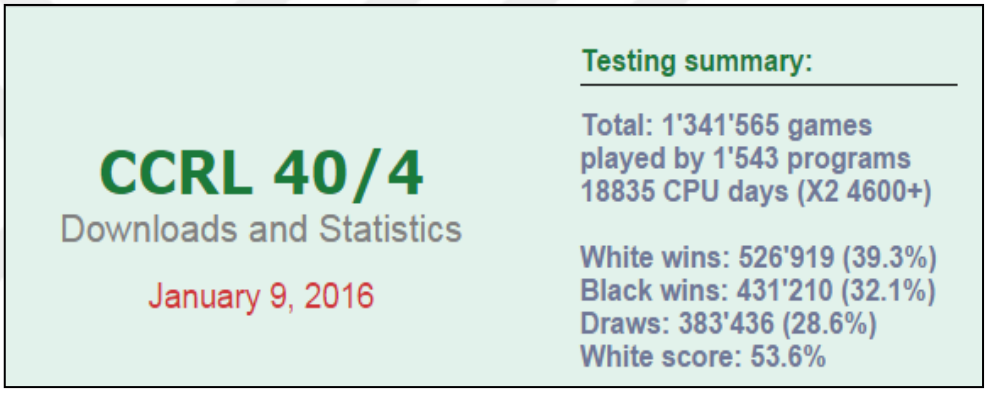
Tablo 3-1 İkili tabanda bitlerin taşıdığı onluk değerler.....55

Tablo 3-2 Taş yeme işlemi bayrağının (MFLAGCAP) on altılı ve ikili sayı tabanlarında gösterimi.....57

1 GİRİŞ

Bu tezle birlikte satranç oyunundaki adaletsizlik giderilmekte ve aynı zamanda eş zamanlı gelişen durumlar/olaylar hakkında karar verme kabiliyetinin gelişmesine katlıda bulunan yeni bir zeka oyunu geliştirilmektedir.

Satranç oyunundaki adaletsizliğin sebebi: Oyuna daima beyaz tarafın başlamasıdır. Bu durum beyaz tarafa her zaman bir hamle avantaj kazandırmaktadır. Eşit güçteki oyuncu çiftleri arasında oynanan satranç karşılaşmaları sonuçlarının analizi incelendiğinde beyaz tarafın siyah tarafa oranla %22,19 daha fazla kazanma oranına sahip olduğunu görülmektedir (Banks G. 2016). Bahse konu istatistik sonuçları Şekil 1-1'de görülmektedir.



Şekil 1-1 www.computerchess.org.uk/ccrl/404/ sitesi test özeti

Toplam 1.341.565 oyun 1.543 bilgisayar programı tarafından oynanmıştır. Bu karşılaşmalardan 526.919'u beyaz, 431.210'u siyah tarafından kazanılmıştır (Şekil 1-1). Beyaz tarafın siyah tarafa göre kazanma oranı;

526.919 / 431.210 = 1,2219'dur, yani beyaz siyaha oranla %22,19 daha fazla oyun kazanmıştır.

İki taraflı oyunların çoğunda ilk hamle avantajının başlayan tarafa üstünlük getirdiği bilinmektedir. Bu üstünlük bir adaletsizlik oluşturmaktadır. Bu çalışmayla birlikte, iki taraflı oyunlar için bu adaletsizliğin giderilebileceği fikri verilmektedir. Ayrıca iki taraflı bir oyun olan satranç oyunu için bu adaletsizliğin giderilebilir olduğu "Eş Zamanlı Satranç Oyunu Uygulaması" ile gösterilmektedir.

İki taraflı oyunlardaki adaletsizlik düşüncesinden farklı olarak, gerçek hayatta olaylar satranç oyunundaki gibi sıralı olarak değil, genellikle eş zamanlı olarak da gerçekleşmektedir. Bazen rakibin hamlesi bilinmeden ve rakiple aynı anda bir hamle yapılması gerekebilmektedir. Eş zamanlı satranç oyunu bu tür durumlar için karar verme yeteneğini geliştirmektedir. Örneğin bir savaş esnasında "önce düşman ateş etsin sonra ben ateş ederim" gibi bir düşünce olamaz. Kararlar eş zamanlı alınmak ve hareketler de eş zamanlı olarak yapılmak zorundadır. Bu durumda aynı anda karşılıklı zayiat verilebilmekte ve/veya avantaj sağlanabilmektedir. Aksi takdirde bekleyen tarafın kaybetme ihtimali çok yüksektir. Örnekler siyaset ve ekonomi gibi hayatın diğer alanlarına da genişletilebilmektedir.

1.1 Eş Zamanlı Satranç Oyunu Nedir?

Bilindiği üzere klasik satranç oyunu sıralı bir oyundur. Taraflar birbirleri hakkında tam bilgiye sahiptir, yani her iki taraf da satranç tahtasında hangi taşların olduğunu ve mevcut durumun ne olduğunu görebilmektedir. Oyuna beyaz taraf başlamakta ve ardından oyun sonuna kadar sıralı olarak hamleler yapılmaktadır. Taraflar birbirinin yaptığı hamleden haberdardır. Bir taraf hamlesini yaptıktan sonra diğer taraf yapılan hamleye karşı bir savunma veya saldırı hamlesi yapmaktadır. Eş zamanlı satranç oyununda da taraflar birbirleri hakkında tam bilgiye sahiptir fakat klasik/sıralı satranç oyununun aksine taraflar birbirinin o anki hamlesinden haberdar değildir. Çünkü taraflar mevcut tahta durumuna göre karar vermekte ve karşılıklı olarak aynı anda hamle yapmaktadırlar.

Eş zamanlı satranç oyunundaki temel kural: Tarafların aynı anda eş zamanlı olarak hamle yapmasıdır. Eş zamanlı hamle: Tarafların da yapacağı hamleye karar verdikten sonra bu hamlelerin kayıt altına alınması ve ardından hamlenin yapılmasıdır. Eş zamanlı satrancın fiziksel bir satranç tahtasıyla gerçek ortamda oynanması yapılacak olan hamlelerin gizli olarak not edilmesi ve karşılıklı hamleler yapıldıktan sonra not edilen hamlelerin gösterilmesi ile gerçekleşebilmektedir. Bu durum biraz zahmetli olduğundan dolayı eş zamanlı

satranç oyunu sanal platformlar için daha uygun bir oyundur. Bunun için de bu çalışmada web ortamı kullanılarak bir oyun platformu geliştirilmiştir. Bu uygulama bilgisayara karşı (insan-bilgisayar modeli[1]) veya gerçek bir rakibe karşı (insan - insan modeli) oynama seçenekleri sunmaktadır.

Daha eşit şartlarda oynanan bir oyun geliştirme fikri sonucunda geliştirilen yeni oyun kurallarından dolayı klasik satrançta uygulanan stratejiler de geçerliliğini büyük oranda yitirmektedir. Dolayısıyla eş zamanlı satranç oyunu için yeni oyun stratejilerinin geliştirilmesi zorunlu hale gelmektedir. Bu durumda eş zamanlı satranç oyunu tarafların birbirinin hamlesinden habersiz olarak aynı anda hamle yapma temeli üzerine inşa edilmiş yeni bir zeka oyunu olarak karşımıza çıkmaktadır.

1.2 Tez Çalışmasının Amacı

Çalışmanın öncelikli amacı eş zamanlı satranç oyunu için belirlenen kurallar setinin tutarlılığını test etmek, tutarsızlık varsa kurallar setini bu tutarsızlığı ortadan kaldıracak şekilde yeniden düzenlemek suretiyle tutarlılığı kanıtlanmış bir kurallar setine ulaşmaktır. Bu amaca ulaşmak için test ortamı olarak bir web uygulaması geliştirilmesi hedeflenmiştir. Bu web uygulaması ile gerçek zamanlı insan – insan ve insan - bilgisayar modellerinin gerçekleştirilmesi amaçlanmıştır.

1.3 Tez Çalışmasının Önemi

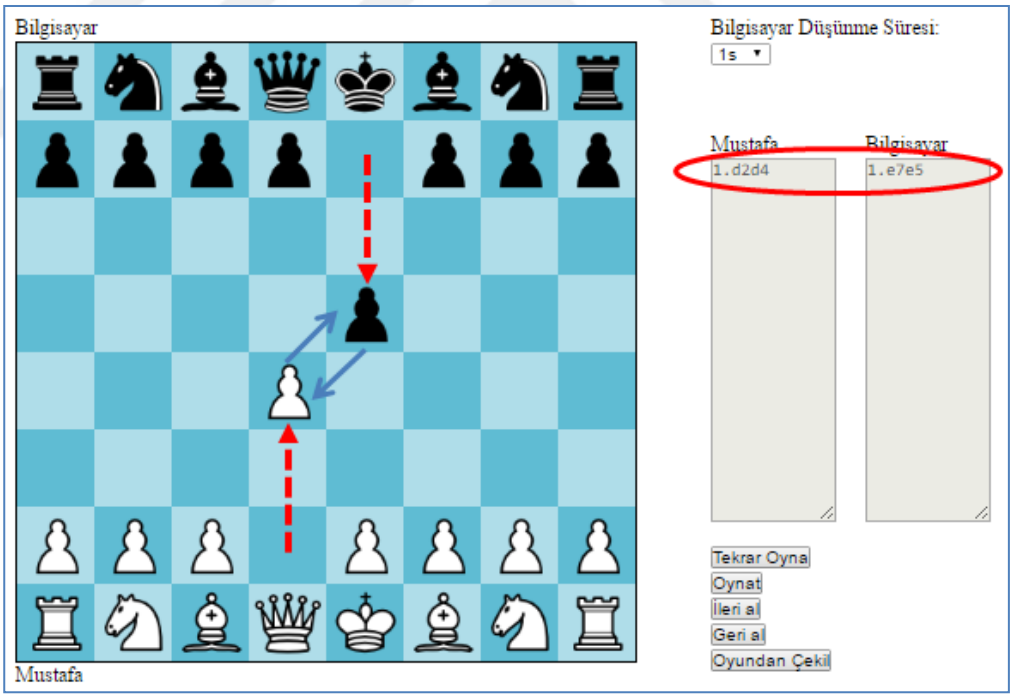
Klasik satranç oyunu dünyada en fazla bilinen ve oynanan, öğrencilerin zihinsel gelişimine yapacağı katkılar düşünülerek okullarda ders olarak okutulan en önemli oyunlardan birisidir. Bu çalışma ile satranç alanında daha kapsamlı düşünmeyi gerektiren yeni bir yaklaşım ortaya konmaktadır. Bu yaklaşım sayesinde klasik satranç ile benzer temel kurallar üzerine inşa edilmiş olan fakat farklı oyun stratejileri geliştirmeyi ve kullanmayı gerektirecek yeni bir oyun platformu oluşturulmaktadır. Bu bağlamda ülkemizde geliştirilen bir oyun

[1] İnsan-bilgisayar modunda bilgisayar sıralı satranç oyununa göre arama yaparak hamlesini belirlemektedir.

platformu (motoru) olarak, oyun motoru geliştirme çalışmalarına olduğu kadar ülke tanıtımımıza da olumlu etki yapabilecek bir çalışmadır.

1.4 Eş Zamanlı Satranç Oyununun Kuralları

Eş Zamanlı Hamle Kuralı: Hamleleri Siyah ve Beyaz taraf eşzamanlı olarak (aynı anda) yaparlar. Yani, her hamle aslında biri beyaz tarafın diğeri siyah tarafın aynı anda gerçekleştirdiği bir hamle çiftidir. Taraflar n. hamlelerini yaparken (n > 0), n-1. hamle sonundaki mevcut tahta durumunu esas almaktadırlar (İlk (n = 1.) hamle için tahtanın başlangıç konumu (n = 0) dikkate alınmaktadır.). Mevcut durumda beyaz taraf ve siyah taraf sıra kendisinde olsa yapabileceği (FIDE Satranç kurallarına uygun, geçerli) her hamleyi eşzamanlı hamle olarak yapabilmektedir. Şekil 1-2'de başlangıç pozisyonundan her iki tarafın açılış hamlesi yaptıktan sonraki tahta durumu görülmektedir. Tarafların eş zamanlı olarak yaptıkları açılış hamleleri kesik çizgili kırmızı oklarla gösterilmekte, bir sonraki hamlede yapacakları hamleler de mavi renkteki oklarla gösterilmektedir.



Şekil 1-2 Zamanlı Hamle Kuralı Örnek I

Şekil 1-3'de ise tarafların karşılıklı olarak birbirini yeme hamlesi yaptığı ve bunun sonucu olarak her iki taşın da tahtadan çıktığı pozisyon görülmektedir. Beyaz piyon d4 karesinden e5 karesine giderek siyah piyonu tahtadan çıkarmış,

aynı anda siyah piyon da e5 karesinden d4 karesine giderek beyaz piyonu tahtadan çıkarmıştır.



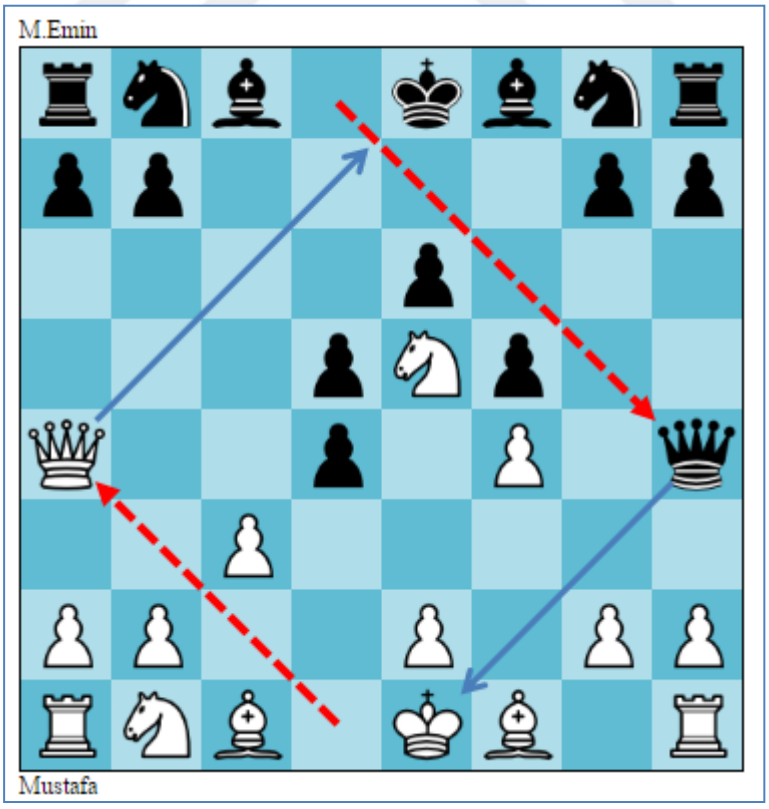
Şekil 1-3 Eş Zamanlı Hamle Kuralı Örnek II

Eş Zamanlı Şah Çekme Kuralı: İki tarafın birinci kurala uygun olarak yaptıkları bir eşzamanlı hamle çifti sonucu, taraflar karşılıklı (ve eşzamanlı) olarak birbirlerine Şah çekebilmektedirler. Şah çekilen taraf(lar), izleyen hamlede FIDE Satranç kurallarına uygun olarak savunma yapmaktadırlar. Karşılıklı Şah çekme hamlesi sonucu, iki taraf karşılıklı (aynı zamanda) Mat olursa oyun berabere sonuçlanmaktadır.

Şekil 1-4'de eş zamanlı şah çekme durumu öncesindeki tahta pozisyonu, Şekil 1-5'de de beyaz vezirin d1 karesinden a4 karesine, siyah vezirin de d8 karesinden h4 karesine hamle yaparak eş zamanlı şah çektikleri görülmektedir. Şekil 1-6 ise beyaz piyonun g2 karesinden g3 karesine, siyah filin de c8 karesinden d7 karesine hamle yaparak eş zamanlı savunma yaptıkları görülmektedir.



Şekil 1-4 Eş Zamanlı Şah Çekme Durumu Öncesindeki Tahta Pozisyonu

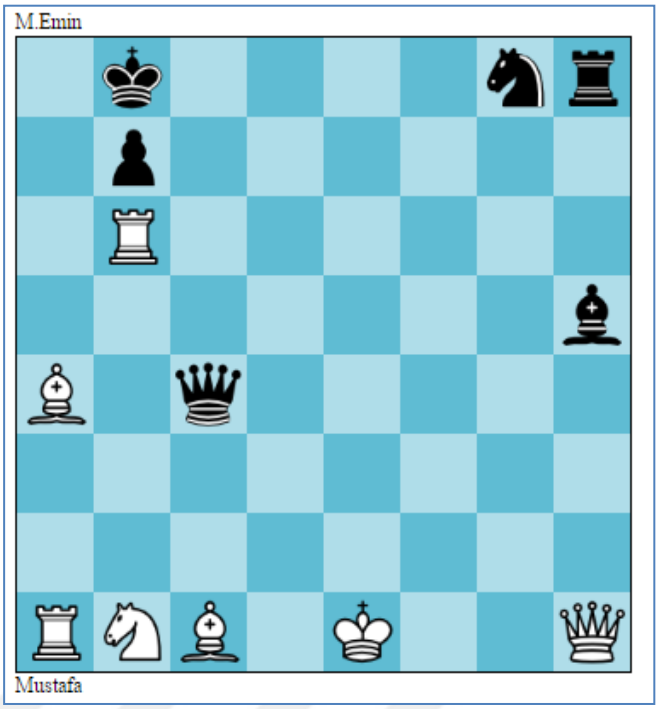


Şekil 1-5 Eş Zamanlı Şah Çekme Durumu

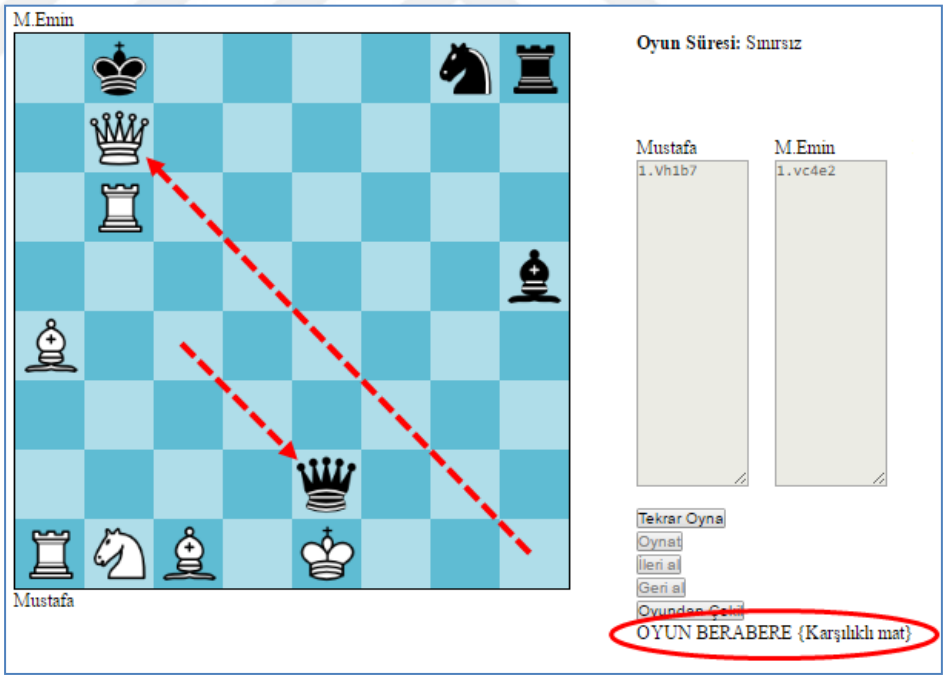


Şekil 1-6 Eş Zamanlı Savunma Durumu

Eş zamanlı mat durumu nadir olarak gerçekleşebilecek durumlardan biri olduğu için farklı bir senaryo ile Şekil 1-7 ve Şekil 1-8'de ele alınmaktadır. Şekil 1-7'de eş zamanlı mat öncesindeki tahta durumu görülmekte, Şekil 1-8'de ise her iki tarafında birbirini karşılıklı olarak mat ettiği görülmektedir.



Şekil 1-7 Eş Zamanlı Mat Öncesi Tahta Durumu

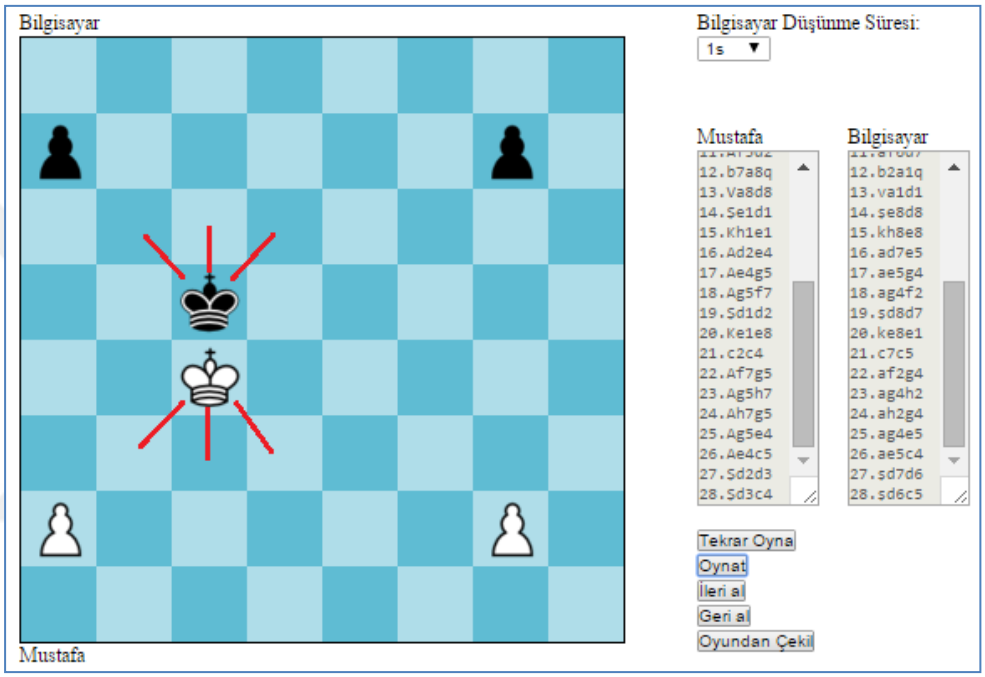


Şekil 1-8 Eş Zamanlı Mat Durumu

Şahların Yan Yana Gelmesiyle İlgili Kural: İki tarafın birinci kurala uygun olarak yaptıkları bir eşzamanlı hamle çifti sonucu, iki şah birbirlerine komşu karelere hareket edip birbirlerini tehdit eder konuma gelebilmektedirler. Bir eş zamanlı oyunda k. hamlede böyle bir durum oluşması halinde k+1. hamlede

her iki şah da tehdit pozisyonundan çıkmak üzere hamle yapmak zorundadır. Eğer Şahlardan biri tehdit pozisyonundan çıkmak için geçerli bir hamle yapamıyorsa Mat olmaktadır. Her iki Şah da tehdit pozisyonundan çıkamıyorsa oyun berabere bitmektedir.

Şekil 1-9'da Şahların komşu karelere düşmesi durumu görülmektedir. Bir sonraki hamlede şahların hareket edebileceği kareler kırmızı çizgilerle gösterilmektedir.



Şekil 1-9 Şahların Komşu Karelere Düşmesi

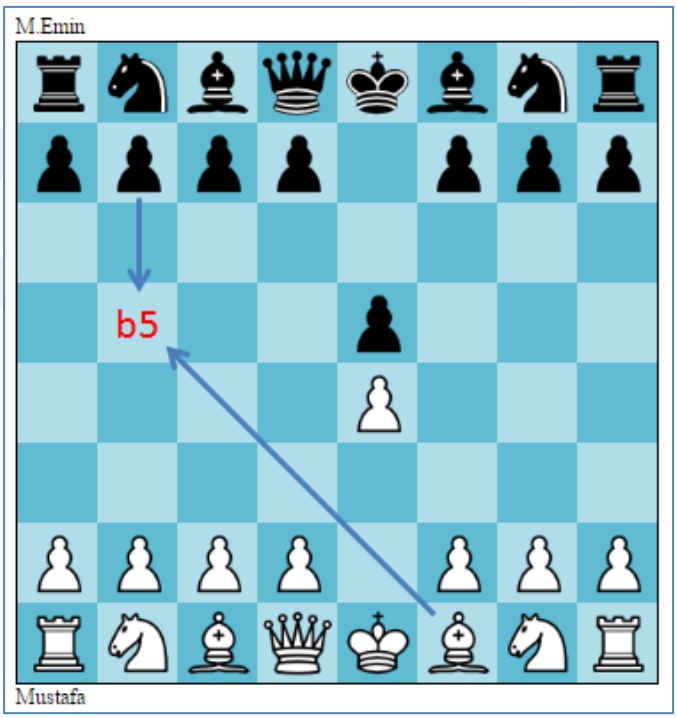
İki Taşın Aynı Kareye Hareket Etmesiyle İlgili Kural: Siyah ve Beyaz'ın eşzamanlı olarak yapacakları bir hamle sonucu iki taş satranç tahtasının aynı karesine gelirse bu taşların ikisi de oyun dışı kalmaktadır (karşılıklı olarak birbirlerini yerler). Aynı kareye gelen taşlardan birisi şah diğeri piyon ise, FIDE kuralları gereği şah tahtadan çıkmadığı için sadece piyon tahtadan çıkarılmaktadır. (Oyun kuralları gereği şahın, piyon haricinde başka bir taşla aynı kareye düşme ihtimali bulunmamaktadır.)

Bu kuralın ilk cümlesinden sonraki kısım, ilk olarak aşağıdaki şekilde belirlenmişti. Ancak bu şekilde etkin olmadığı düşünülerek değiştirilmiştir. Şayet aşağıda belirtildiği gibi hamlelerin karşılıklı olarak geri alınma kuralı korunmuş olsaydı, dezavantajlı olan tarafa üçlü tekrar kuralını kullanma şansı doğacaktı. Bu

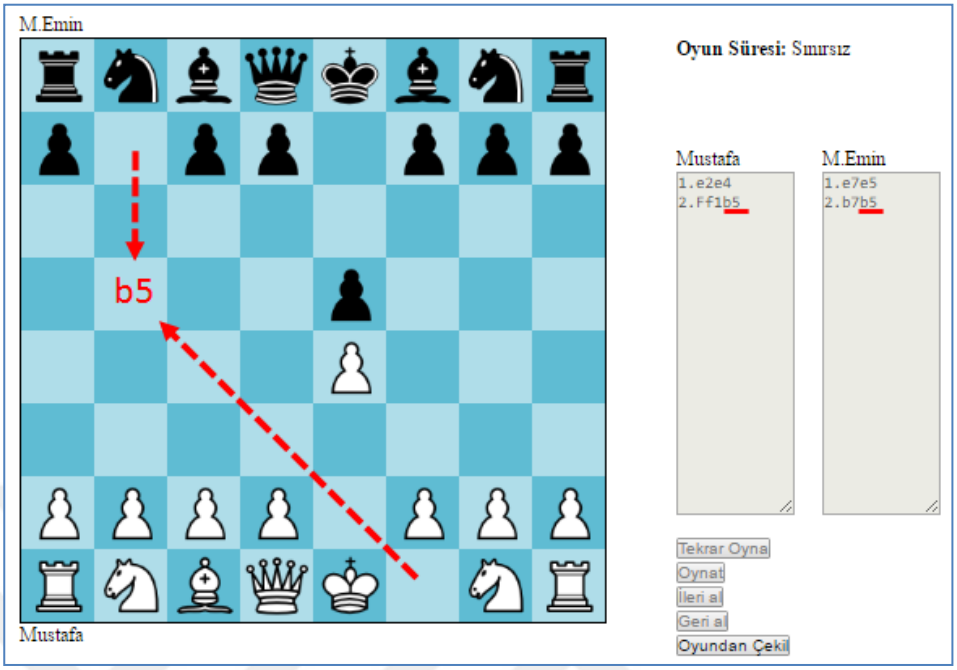
durumun oyun dengesini bozacağı düşünülerek kural değişikliği yoluna gidilmiştir.

"Aynı kareye gelen taşlardan birisi şah ise hamle karşılıklı olarak geri alınır. Geri almadan bir sonraki hamle çiftinde taraflar farklı hamleler yapabilecekleri gibi yine aynı kareye hamle yapabilirler. Bu durum toplam üç kez tekrarlanırsa oyun Pat olur."

Şekil 1-10'da aynı kareye (b5) hareket edecek olan taşlar gösterilmektedir. Şekil 1-11'de ise taşların hareketi sonucu ikisinin de aynı anda tahtadan çıktığı durum görülmektedir.



Şekil 1-10 Aynı Kareye Hareket Etme



Şekil 1-11 Aynı Kareye Hareket Sonucu

Tutsak Taş Kuralı: Oyunun n. hamlesinde rakip bir taş tarafından tehdit edilen bir taş (TT: Tutsak Taş), n+1. hamlede birinci kurala uygun her hamleyi yapabilmektedir. Örneğin n. hamlede tehdit edilen bir piyon n+1. hamlede; çaprazındaki rakip taşı yiyebilmekte, ileri doğru hamle yapıp rakip bir taşı tehdit edebilmekte, kendi renginde bir taşı koruyacak bir kareye gelebilmekte ve/veya Şah çekebilmektedir. n. hamlede tehdit ettiği taşı (TT'ı) n+1. hamlede almaya karar veren rakip bunu n+1. hamlede o taşın (TT'nin) konumundan ya da hamlesinden bağımsız olarak yapabilmektedir. Yani, TT'ın n+1. hamlede hareket ettirilmiş olması rakibin n+1. hamlesinde onu almasına engel değildir. Bununla birlikte TT'ın eğer varsa n+1. hamlede gerçekleştirdiği taş yeme eylemi de geçerlidir.

Şekil 1-12'de beyaz vezir, siyah vezir, siyah kale, siyah at ve siyah fili tutsak taş durumuna düşürmüştür. Aynı zamanda siyah kale,siyah vezir ve siyah fil de beyaz veziri tutsak taş durumuna düşürmüştür. Taşların n+1. hamlede yapacağı hareketler oklarla gösterilmektedir. Şekil 1-13'de de n+1. hamle sonucu oluşan tahta durumu görülmektedir.



Şekil 1-12 Tutsak Taş n. Hamle



Şekil 1-13 Tutsak Taş n+1. Hamle

1.5 Eş Zamanlı Satranç Oyununun Tarihçesi

Eş Zamanlı Satranç Oyununun tanımı ve kuralları ilk olarak 15 Eylül 2014 tarihinde Prof.Dr.Mehmet Emin DALKILIÇ tarafından ortaya konmuştur.

17 Eylül 2014 tarihinde ise bu yeni oyun ile ilgili olarak kuralların tutarlılığını test etmek amacıyla bir tez çalışması kapsamında web tabanlı bir uygulama geliştirme çalışmasına başlanmıştır.

23 Mart 2015 tarihinde bu uygulamanın İnsan-Bilgisayar modeli tamamlanmıştır.

26 Mayıs 2015 tarihinde uygulamanın ilk sözlü tanıtımı UBE'de bir seminer dersi sunumu ile yapılmıştır.

27 Aralık 2015 tarihinde İnsan-İnsan modelinin tamamlanmasının ardından Internet'te "http://schess.herokuapp.com/" isimli bir web sayfası üzerinde uygulama yayınlanmaya başlanmıştır.

2 KLASİK SATRANÇ PROGRAMLAMA VE EŞ ZAMANLI SATRANÇ PROGRAMLAMA

Bu tez çalışmasında halihazırda kullanılan satranç programları incelenmiş ve mevcut açık kaynak kodlarından faydalanılmıştır. Bu kapsamda bir adet Java programlama diliyle kodlanmış basit düzeyde bir satranç programı (Warkentin-I 2014), bir adet yine Java programlama diliyle kodlanmış ileri düzeyde bir satranç programı (Warkentin-II 2014), bir adet C programlama diliyle kodlanmış (Bluefever-I 2014) ve bir adet de Java Script (Bluefever-II 2014) programlama diliyle kodlanmış satranç programı incelenmiştir. İlk üç uygulamaya sonradan entegre edilen klasik/sıralı satranç oyunu için yazılmış GUI (Graphical User Interface - Grafik Kullanıcı Ara Yüzü)'ler kullanılmaktadır. Bu GUI'ler satranç motorlarıyla XBoard ve WinBoard protokolleri aracılığıyla haberleşmektedirler (chess-programming-I 2016). Mevcut GUI'lerle satranç motorları arasındaki iletişim klasik satranca uygun olarak en basit şekliyle bir tarafın hamlesini alıp onu tahtada göstermekte ve diğer tarafın hamlesini alıp onu da karşı tarafın satranç motoruna iletmektedir (Mann 2009). XBoard ve WinBoard protokolleri ile kullanılan GUI'ler her ne kadar klasik satranç oyunu için oldukça avantajlı uygulamalar olsa da eş zamanlı satranç oyunu için kullanışlı değillerdir. Çünkü eş zamanlı satrançta her iki tarafında hamlesi alındıktan sonra aynı anda satranç tahtasına işlenmesi gerekmektedir fakat mevcut GUI'lerde böyle bir kabiliyet bulunmamaktadır. Hem bu nedenle hem de web ara yüzü entegrasyonu olmadığı için ilk üç satranç uygulaması bu tez çalışmasında kullanılmamıştır. Java Script ile kodlanmış olan satranç motoru uygulaması hem web ara yüzlü olduğundan hem de grafik ara yüzüne müdahale edilip değiştirilebildiğinden dolayı prototip düzeyde bir çalışmaya uygun bir kaynak kod olduğu görülmüş ve tez çalışması bu kodların eş zamanlı satranç oyununa uyarlanmasıyla gerçekleştirilmiştir.

Uygulama istemci tarafı ve sunucu tarafı olmak üzere iki ana kısımdan oluşmaktadır. Hamlelerin alınması (insan-insan veya insan-bilgisayar modeli), bilgisayarın en iyi hamleyi araması (insan-bilgisayar modeli) ve hamlelerin işleme konulması istemci tarafında yapılmaktadır. Sunucu tarafında ise uygulama bir web ara yüzü aracılığıyla kullanıma sunulmakta ve oyuncuların hamleleri, oyun sonuçları ve zaman bilgisi karşılıklı olarak oyunculara iletilmektedir.

2.1 Satranç Programlamaya Genel Bakış

Satranç oyunu kodlamanın temelinde oyun tahtasının bilgisayardaki temsili yatmaktadır. Bununla hem tahta durumu hem de oyunun kuralları takip edilmektedir. Bu kurallar: Bilgisayarda temsili oldukça karışık olabilen rok kuralı, tekrarlama kuralları, geçerken alma (en passant) kuralı gibi bir çok satranç kuralını kapsamaktadır (chess-programming-II 2016).

Satranç oyunu kodlamada diğer önemli bir husus ise bilgisayarın iyi bir hamle seçmesini sağlayan "arama"(search) ve "değerlendirme"(evaluation) kısmıdır (chess-programming-II 2016).

Son olarak da tutulan tahta bilgilerinin, hamleler ve sonuçlarının görsel olarak gösteriminin yapıldığı GUI'ler (Graphical User Interface - Grafik Kullanıcı Ara Yüzü) ile ilgili işlemler yer almaktadır (chess-programming-II 2016).

2.2 Satranç Oyun Tahtasının Bilgisayarda Temsili

Satranç programlamada, oyun taşı merkezli (Piece Centric) ve oyun karesi merkezli (Square Centric) olarak temelde iki farklı tahta temsil yöntemi bulunmaktadır. Oyun taşı merkezli yöntemlerde tahtadaki taşların bir listesi o taşların işgal ettikleri kare bilgisi ile birlikte tutulur. Oyun karesi merkezli yöntemde ise bir karenin boş mu yoksa belirli bir taş tarafından işgal edilip edilmediği bilgisi tutulur. Eş zamanlı satranç oyunu için referans alınan kodlamada bu iki yöntemden de faydalanılarak melez bir çözüm üretilmiştir (Bluefever-II 2014).

Öncelikli olarak tahtadaki tüm kareleri temsil eden 64 elemanlı bir dizi oluşturulmuştur. Bu dizi 0'dan 63'e kadar indis numarası olan ve Şekil 2-1'deki gibi düşünülmüş bir dizidir. Burada her bir indis numarası bir karenin içerisinde görülmektedir. Şekil 2-1'de ayrıca satranç tahtasının satır ve sütun numaralandırması da görülmektedir.

1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
3	16	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30	31
5	32	33	34	35	36	37	38	39
6	40	41	42	43	44	45	46	47
7	48	49	50	51	52	53	54	55
8	56	57	58	59	60	61	62	63
	a	b	c	d	e	f	g	h

Şekil 2-1 Satranç Tahtası Temsili: 64 Elemanlı Dizi

Bu dizinin ilk elamanı (0. indis) A1 karesini son elemanı (63. indis) ise H8 karesini temsil etmektedir. Diğer elemanlarda sırasıyla diğer kareleri temsil etmektedir. Bu durum Şekil 2-2'de görülmektedir.

1	a1	b1	c1	d1	e1	f1	g1	h1
2	a2	b2	c2	d2	e2	f2	g2	h2
3	a3	b3	c3	d3	e3	f3	g3	h3
4	a4	b4	c4	d4	e4	f4	g4	h4
5	a5	b5	c5	d5	e5	f5	g5	h5
6	a6	b6	c6	d6	e6	f6	g6	h6
7	a7	b7	c7	d7	e7	f7	g7	h7
8	a8	b8	c8	d8	e8	f8	g8	h8
	a	b	c	d	e	f	g	h

Şekil 2-2 Satranç Tahtası Temsili: Kare, İndis İlişkisi

Kullanılan programlama yönteminde her bir satranç taş türü farklı bir rakam ile aşağıdaki gibi temsil edilmektedir;

- Boş Kare (EMPTY): 0
- Beyaz Piyon (White Pawn - wP): 1
- Beyaz At (White Knight - wN) : 2
- Beyaz Fil (White Bishop - wB) : 3
- Beyaz Kale (White Rook - wR) : 4
- Beyaz Vezir (White Queen - wQ) : 5
- Beyaz Şah (White King - wK): 6
- Siyah Piyon (Black Pawn - bP): 7

- Siyah At (Black Knight - bN) : 8
- Siyah Fil (Black Bishop - bB) : 9
- Siyah Kale (Black Rook - bR) : 10
- Siyah Vezir (Black Queen - bQ) : 11
- Siyah Şah (Black King - bK): 12

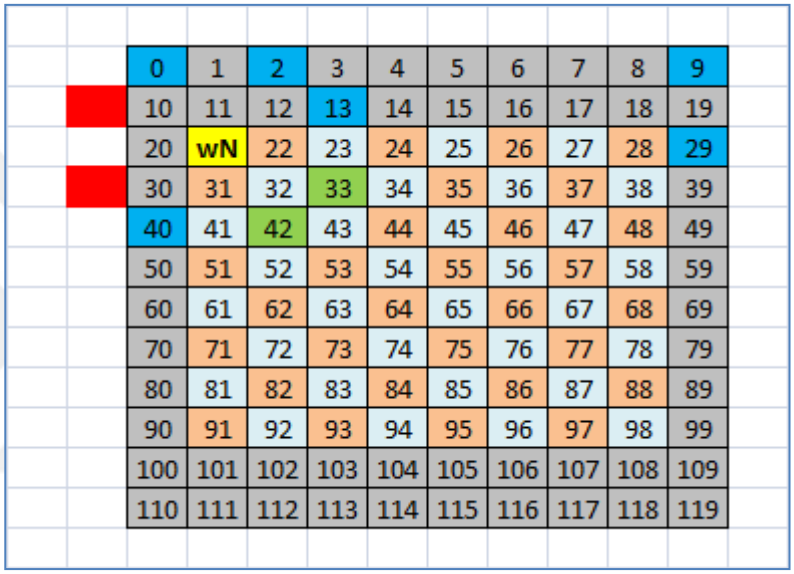
Şekil 2-3'de taşların tahta üzerine yerleşimi görülmektedir.

1	4	2	3	6	5	3	2	4	Beyaz
2	1	1	1	1	1	1	1	1	Taşlar
3	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	
7	7	7	7	7	7	7	7	7	Siyah
8	10	8	9	12	11	9	8	10	Taşlar
	a	b	c	d	e	f	g	h	

Şekil 2-3 Satranç Tahtası Temsili: Taşlar ve Yerleşimi

Her ne kadar satranç tahtasının ve taşların uygulamada nasıl temsil edileceği belirlenmiş olsa da bir taşın yapabileceği hamlelerin çıkarılması için gereksiz ve hata oranı yüksek olabilecek kodlamalardan kaçınmak maksadıyla 120 eleman uzunluğunda bir dizi kullanılarak satranç tahtası temsil edilmiştir. Bunun nedeni ise yapılabilecek hamlelerin bulunması için bir döngü kullanıldığında kontrol edilen bölgenin hala satranç tahtasının içerisinde mi yoksa dışarısında mı olduğunun kontrolünü yapmaktır. Şekil 2-4'de 120 elemanlı bir dizinin satranç tahtası olarak uyarlanması görülmektedir. Gri renk ile gösterilen alanlar tahta dışı (OFF BOARD) olarak işaretlenen alanlardır. A1 karesi 21. indis, H8 karesi ise 98. indis olarak belirlenmiştir. Bu 120 elemanlı dizinin kullanılma amacını bir örnekle de açıklayalım; Bir at tahta üzerinde "L" şeklinde hareket eder. Bu nedenle satranç tahtasının üstüne ve altına iki sıra boş kare yerleştirilmiştir. Bunun nedeni son sıradaki at iki ileri ve bir yana hamle yaptığında dizinin dışına çıkmasını engel olmak içindir. Tahtanın sağ ve sol sütunlarında sadece bir sütun boş kare olmasının nedeni ise o sütunların birbirinin devamı olmasıdır. Nihayetinde bir dizi içerisinde hareket edildiği ve Şekil 2-4'deki gibi 21. karede (A1) bir Beyaz At (wN) olduğu düşünelim. wN bir kare yukarı ve iki kare sola gitmek istediğinde her ne kadar 10. karenin solundaki kırmızı ile gösterilen alana gidiyor gibi

görünse de aslında dizi içerisindeki 9. kareye hareket ediyor. Dolayısıyla dizi dışına çıkılmamış oluyor ve hareket edilmek istenen kare tahta dışı (OFF BOARD) olduğu için geçerli bir hamle olarak kaydedilmiyor. At taşı için kodlama esnasında 8, 12, 19 ve 21 sayıları taşın bulunduğu kareden çıkarılarak ve taşın bulunduğu kareye ilave edilerek bir atın hamle yapabileceği kareler tespit edilmektedir. Bu örnekte de wN'ın sadece yeşil renkle gösterilen 33. ve 42. karelere hareket edebileceği ve diğer karelerin tahta dışı olduğu görülmektedir. Kırmızı renkle gösterilen kısım ise dizinin dışarısında bir alandır ve yapılabilecek hamlelerin tespiti için yapılan döngü esnasında o alanlara çıkmamaktadır.



Şekil 2-4 120 Elemanlı Dizi ve Hamlelerin Tespiti

2.3 Takip Edilmesi Gereken Bilgiler

Satranç oyunun bir bütün olarak bilgisayarda temsil edilebilmesi için oyun tahtasının temsilinin yanında oyun ile ilgili diğer bilgi ve kuralların da takip edilmesi gerekmektedir. Bu kuralların takibi için kullanılan başlıca değişkenler aşağıdadır;

- **Oyun Tahtası:** GameBoard.pieces = new Array (120);

- **Oyun sırası bilgisi :** GameBoard.side = COLOURS.WHITE;
- **50 hareket kuralı :** GameBoard.fiftyMove = 0;
- **Kaçıncı yarım hamle[1] :** GameBoard.hisPly = 0;
- **Önceki pozisyon ve hamle bilgileri :** GameBoard.history = [];
- **Mevcut durumda Alpha Beta Search'te kaçıncı derinlik (**Arama ağacında yapılan yarım hamle sayısı.**):** GameBoard.ply = 0;
- **Geçerken Alma (enPassant) Karesi:** GameBoard.enPas = 0;
- **Rok izni bilgisi:** GameBoard.castlePerm = 0;
- **Materyal bilgisi:** GameBoard.material = new Array (2);
- **Hangi taştan kaç tane olduğu :** GameBoard.pceNum = new Array (13);
- **Taşların hangi karelerde olduğu:** GameBoard.pList = new Array (14 * 10);
- **Pozisyon anahtarı:** GameBoard.posKey = 0;

2.4 Bazı Önemli Değişkenler ve Sabitler

Bu kısımda programın tamamını ilgilendiren değişken ve sabitlerin ne amaçla kullanıldığı bilgisi verilmektedir. Programın tamamını ilgilendirmeyen değişken ve sabitler, o alanla ilgili kodlamanın anlatıldığı yerde ele alınmaktadır.

Oyun taşları, kodlamanın daha okunur olması için aşağıdaki şekilde tanımlanmıştır,

var PIECES = { EMPTY : 0, wP : 1, wN : 2, wB : 3,wR : 4, wQ : 5, wK : 6, bP : 7, bN : 8, bB : 9, bR : 10, bQ : 11, bK : 12 };

Boş kareler kodlamada esnasında "0" yerine PIECES.EMPTY kullanılarak gösterilmiş böylelikle kod okunurluğu arttırılmıştır. Diğer taşların temsil edildiği alanlar da benzer şekilde kodlanmıştır. Örneğin; Beyaz Piyon, "1" yerine PIECES.wP olarak tanımlanmıştır.

[1] Klasik satranç oyununda her bir tarafın yaptığı hamle yarım hamle olarak tanımlanmaktadır. Hem siyah hem de beyaz hamlesini tanımladığında bir tam hamle yapılmış olmaktadır. Eş zamanlı satranç oyununda ise aynı değişkenle tam hamle sayısı tutulmaktadır. Çünkü her iki taraf da aynı anda hamle yapmaktadır.

Tahtadaki kare sayısı, var BRD_SQ_NUM = 120;

Tahtadaki Sütunlar,

var FILES = { FILE_A:0, FILE_B:1, FILE_C:2, FILE_D:3, FILE_E:4, FILE_F:5, FILE_G:6, FILE_H:7, FILE_NONE:8 };

Tahtadaki Satırlar,

var RANKS = { RANK_1:0, RANK_2:1, RANK_3:2, RANK_4:3, RANK_5:4, RANK_6:5, RANK_7:6, RANK_8:7, RANK_NONE:8 };

Renkler (taraflar), var COLOURS = { WHITE:0, BLACK:1, BOTH:2 };

Bazı anahtar kareler, var SQUARES = { A1:21, B1:22, C1:23, D1:24, E1:25, F1:26, G1:27, H1:28, A8:91, B8:92, C8:93, D8:94, E8:95, F8:96, G8:97, H8:98, NO_SQ:99, OFFBOARD:100};

Yukarıda tanımlanmış olan anahtar karelerden A1, B1, C1, D1, E1, F1, G1, H1, A8, B8, C8, D8, E8, F8, G8, H8 kareleri rok ile ilgili karelerdir. NO_SQ tanımlaması, herhangi bir kare bilgisinin olmadığını göstermektedir. Örneğin; oyun esnasında bir geçerken alma karesi oluşmamışsa, geçerken alma karesi bilgisini tutan değişkene NO_SQ değeri atanmaktadır. OFFBOARD tanımlaması ise asıl oyun tahtasını oluşturan 64 karenin dışını tanımlamaktadır. Bu tanımlamalar, kodlama esnasında bazı anahtar karelerin numaraları yerine isimlerinin girilmesi için yapılmış tanımlamalardır.

BOOLEAN, var BOOL = { FALSE:0, TRUE:1 };

3 EŞ ZAMANLI SATRANÇ OYUN MOTORU KODLAMA

Bu bölümde grafik ara yüzle ilgili olmayan, asıl oyun motorunu oluşturan kodlar ele alınmaktadır. Grafik kullanıcı ara yüzü ile ilgili kısım ise 4. bölüm içerisinde yer almaktadır. Kodların incelenmesi genel itibariyle fonksiyon temelli olarak yapılmaktadır.

Eş zamanlı satranç uygulaması web ara yüzlü bir uygulamadır ve sunucu tarafının kodlanmasında JavaScript kütüphanelerinden birisi olan node.js kütüphanesi kullanılmıştır. node.js kütüphanesine yardımcı olarak da express.js ve socket.io kütüphaneleri de kullanılmıştır. Kodları çalıştıracak olan uygulama (node) sunucu üzerinde başlatıldığında ilk olarak kodların yer aldığı klasörün kök dizinindeki server.js dosyasını çalıştırmaktadır (Bkz. 3.1). server.js içerisinde istemciye yanıt olarak gönderilecek olan index.ejs dosyası çağrılmaktadır (Bkz. 4). index.js dosyası uygulamanın karşılama ekranını kullanıcıya sunmaktadır. Bu ekranda kullanıcıya yeni bir oyun kurması veya kurulmuş bir oyuna dahil olması imkanı sunulmaktadır. Aynı zamanda uygulama ile ilgili kurallar, iletişim ve hakkımızda sayfalarına index.ejs dosyası aracılığıyla ulaşılmaktadır. Kullanıcı kendisini karşılayan ekranda oyun kurulumu ile ilgili tercihler yaptığında web sayfasındaki dinamik işlemler index.js dosyası aracılığıyla yapılmaktadır (Bkz.4.2). İstemci tarafından "http://schess.herokuapp.com/" sitesi çağrıldığında kullanıcıya index.ejs dosyasında yer alan karşılama ekranın istemci tarafından yüklenmesinin ardından (kullanıcı yeni oyun kurulumu ile ilgili tercihlerinin yaparken) arka planda eş zamanlı satranç oyununun oynanacağı ortam istemci bilgisayarın web tarayıcısına yüklenmektedir (Bkz.3.2). Bu alan oyun tercihleri yapılana kadar gizli tutulduğu için kullanıcı tarafından görülmemektedir. Yeni oyun tercihleri tamamlandığında yapılan tercihler doğrultusunda bu alan kullanıcıya görünür hale getirilmekte ve oyun kurulumunun yapıldığı alan ekrandan kaldırılmaktadır. Oyun kurulumunun ardından oyuncular karşılıklı olarak hamle yapmakta ve bu hamleler karşılıklı olarak oyun ekranlarına yansıtılmaktadır.

3.1 Sunucu ile İlgili Kodlama

Sunucu ile ilgili kodlama server.js dosyası içerisinde yer almaktadır. Bu dosyanın başlangıç kısmı aşağıdaki şekildedir;

```
1 var cc  = require('config-multipaas'),
```

```
2       config  = cc(),
3       express = require('express'),
4       app = express(),
5       morgan = require('morgan'),
6       server = require('http').Server(app),
7       io = require('socket.io').listen(server),
8       bodyParser = require('body-parser'),
9       path = require('path'),
10      COLOURS = { WHITE:0, BLACK:1, BOTH:2 };
11
12  //configure app
13  app.set('view engine', 'ejs');
14  app.set('views', path.join(__dirname, 'views'));
15
16  //use middleware
17  app.use(morgan('dev'));
18  app.use(express.static(__dirname + '/public'));
19
20  server.listen(config.get('PORT'), config.get('IP') , function () {
21    console.log( "Listening on " + config.get('IP') + ", port "+config.get('PORT') )
22  });
23
24  var selectedUserName='',
25      selectedSide='',
26      selectedPlayer=0;
27
28  var allPlayers = [];
29  var waitingList = [];
30
31  app.get('/', function(req, res){
32      res.render('index', {waitingList: waitingList});
33  } );
```

server.js dosyasının 1. ve 9. satırları arasında, uygulama içerisinde kullanılacak olan kütüphaneler çağrılarak ilgili değişkenlere atanmaktadır. 10. satırda server.js içerisinde kullanılacak olan tarafları belirten bir sabit tanımlanmaktadır. 12. ve 13. satırlarda, kullanıcıya sunulacak görsel alandaki dosyanın formatı ve yeri belirlenmektedir. "__dirname" depolama alanı içerisinde uygulamanın bulunduğu yeri göstermektedir. 17. satırda, yapılan http istekleri ile ilgili loglar sunucu konsoluna yazdırılmaktadır. 18. satırda, statik yapıdaki dosyaların yolu belirtilmektedir. 20. ve 22. satırlar arasında, istemci tarafından gelen isteklerin alınabilmesi için sunucu tarafından belirlenen port ve IP

numaraları dinlenmekte ve dinlenen bu port ve IP numaraları konsola yazdırılmaktadır. 24. ve 29. satırlar arasında, oyuncular arasında veri transferi için kullanılan değişkenler tanımlanmaktadır. 31. ve 33. satırlar arasında, istemci tarafından bir istek geldiğinde yani, "http://schess.herokuapp.com/" sitesi bir Internet gezgini tarafından çağrıldığında "/views/index.ejs" dosyası ve waitingList (rakip bekleyen oyuncu listesinin tutulduğu dizin) istemciye gönderilmektedir. server.js dosyasının geri kalan kısımları daha sonra ilgili kısımlar içerisinde ele alınmaktadır. Uygulamanın geneli istemci tarafıyla ilgili olduğu için ilerleyen kısımlarda genellikle istemci tarafında çalışan kodlar ele alınacaktır.

3.2 İstemci Tarafında Uygulamanın Başlatılması

İstemci uygulamaya erişim talebi yaptıktan sonra sunucu tarafından index.ejs dosyası gönderilmekte ve bu dosya istemcinin İnternet gezgininde açılmaktadır. Bu dosyanın yüklenmesi tamamlandıktan sonra main.js içerisinde aşağıda yer alan kod çalıştırılmaktadır.

```
1 $(function() {
2   init();
3   console.log("Main Init Called");
4   NewGame(START_FEN);
5 });
```

Yukarıdaki kodun 2. satırında **init** fonksiyonu çağrılarak uygulamanın başlangıcında kullanılacak olan değişken ve sabitlerin ilklenmesini yapacak olan fonksiyonların çağrılması sağlanmaktadır; **init** fonksiyonu main.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1 function init() {
2       console.log("init() called");
3       InitFilesRanksBrd();
4       InitHashKeys();
5       InitSq120To64();
6       InitBoardVars();
7       InitMvvLva();
8       InitBoardSquares();
9 }
```

init fonksiyonunun 2. satırında, init fonksiyonunun çağrıldığı konsola yazdırılmaktadır. 3. satırında, **InitFilesRanksBrd** fonksiyonu çağrılarak satır ve

sütun sabitlerinin başlangıç değerleri ilklenmektedir (Bkz. 3.4). 4. satırında **InitHashKeys** fonksiyonu çağrılarak pozisyon özetleme anahtarının oluşturulmasında kullanılacak olan rastgele sayılar üretilmektedir (Bkz. 3.6). 5. satırında **InitSq120To64** fonksiyonu çağrılarak 64 ve 120 eleman uzunluğundaki dizinler arasındaki dönüşümler için gerekli ilklemeler yapılmaktadır (Bkz. 3.7). 6. satırda **InitBoardVars** fonksiyonu çağrılarak oyun geçmişindeki pozisyon bilgilerini tutan değişkenler ilklenmektedir (Bkz. 3.19). 7. satırda, arama fonksiyonları ile ilgili fonksiyonlardan birisi olan **InitMvvLva** fonksiyonu çağrılmaktadır. Eş zamanlı satranç oyunu için bir arama fonksiyonu henüz geliştirilmediği için ve bu uygulamadaki mevcut arama fonksiyonları da klasik satranca göre olduğu için arama fonksiyonları bu çalışma kapsamında incelenmemektedir. 8. satırda, oyun tahtasının karelerini oluşturan **InitBoardSquares** fonksiyonu çağrılmaktadır (Bkz. 4.4).

main.js dosyasının 4. satırında **NewGame** fonksiyonu START_FEN parametresi ile çağrılmaktadır.

```
 1 function NewGame(fenStr) {
 2     ParseFen(fenStr);
 3     PrintBoard();
 4     ClearMoveDisplay();
 5     InitBoardVars();
 6     SetInitialBoardPieces();
 7     CheckAndSet();
 8     if (player == 2){
 9         GameBoard.side = playerSide;
10     }else{
11         if(playerSide == COLOURS.WHITE ){
12             GameBoard.side = COLOURS.BLACK;
13             PreSearch();
14             GameBoard.side = COLOURS.WHITE;
15         }else if (playerSide == COLOURS.BLACK){
16             GameBoard.side = COLOURS.WHITE;
17             PreSearch();
18             GameBoard.side = COLOURS.BLACK;
19         }
20     }
21 }
```

NewGame fonksiyonu çağrılırken kullanılan START_FEN parametresi oyun taşlarının başlangıç konumlarının kodlanmış olduğu bir dizindir. 2. satırda, START_FEN dizini (veya parametre olarak verilmiş olan FEN dizini) **ParseFen** fonksiyonuna parametre olarak verilerek **ParseFen** fonksiyonu çağrılmaktadır. **ParseFen** fonksiyonu ile verilen FEN metni taranmakta ve içerisinde kodlanmış olan değerler çözümlenerek ilgili değişkenlere atanmaktadır (Bkz. 3.8). 3. satırda **PrintBoard** fonksiyonu çağrılarak oyun tahtası konsola yazdırılmaktadır (Bkz. 3.10). 4. satırda **ClearMoveDisplay** fonksiyonu çağrılarak grafik ara yüzde oyuncuların hamlelerini gösteren alan ilklenmektedir. 5. satırda, **InitBoardVars** fonksiyonu çağrılarak oyun geçmişindeki pozisyon bilgilerini tutan değişkenler ilklenmektedir (Bkz. 3.19). 6. satırda, **SetInitialBoardPieces** fonksiyonu çağrılarak oyun taşları grafik ara yüz üzerindeki yerlerine yerleştirilmektedir (Bkz. 4.4). 7. satırda **CheckAndSet** fonksiyonu çağrılarak tahtaya işlenmiş olan FEN metni ile verilen oyun pozisyonunda bir şah-mat durumu olup olmadığı kontrol edilmektedir (Bkz. 3.22 ve 3.24). Oyun kurulumu esnasında belirlenmiş olan oyuncu sayısı iki ise (8. satır) GameBoard.side değişkenine oyuncunun belirlemiş olduğu taraf atanmaktadır. Şayet oyuncu sayısı bir olarak belirlenmişse yani, oyuncu bilgisayara karşı oynayacaksa (10. satır) GameBoard.side değişkenine bilgisayarın tarafı atanıp bilgisayarın en iyi hamlesini belirlemesi sağlanmaktadır (13. satır ve 17. satır). Ardından da GameBoard.side değişkenine oyuncunun tarafı atanıp oyuncunun hamle yapmasına imkan verilmektedir (11. ve 20. satırlar). 8. ve 20. satırlar arasındaki kod ilk oyun kurulumunda kullanılamamaktadır. Çünkü **NewGame** fonksiyonu çağrıldığında "player" ve playerSide değişkenlerine bir değer atanmamış durumdadır. Bu alan daha önceden oynanmış bir oyun aynı şartlarda tekrar oynanmak istenirse kullanılmaktadır. İlk oyun kurulumunda değişken değerleri atandıktan sonra benzer bir alan bulunmaktadır (Bkz. 4.2).

3.3 Oyun Tahtasının Tanımlanması

Oyun tahtası board.js dosyası içerisinde tanımlanmaktadır. board.js dosyası içerisinde yer alan oyun tahtasına ait kodların bir kısmı aşağıda gibidir;

```
var GameBoard = {};
GameBoard.pieces = new Array(BRD_SQ_NUM);
GameBoard.side = COLOURS.BOTH;
GameBoard.fiftyMove = 0;
GameBoard.hisPly = 0;
```

```
GameBoard.enPas = [0,0];
GameBoard.history = [];
GameBoard.ply = 0;
GameBoard.castlePerm = 0;
GameBoard.material = new Array(2);
```

"GameBoard" isimli JavaScript nesnesi (object) oyundaki mevcut tahta durumunu takip amacıyla tutulmaktadır.

"GameBoard.pieces" nesne özelliği (object property) bir dizindir (array). Bu dizin ile oyun tahtasının kareleri ve üzerindeki taş bilgisi tutulmaktadır. Örneğin; beyaz kale (PIECES.wR veya 4) tahta üzerinde A1 karesi üzerinde yer alıyor ise GameBoard.pieces dizininde 21 numaralı indise 4 değeri atanmaktadır.

"GameBoard.side" nesne özelliği ile oyun sırasının hangi tarafta olduğu belirlenmektedir. Her ne kadar eş zamanlı satranç oyununda iki taraf da aynı anda hareket etse ve sıra kavramı kalmamış olsa da bir tarafın yapılabileceği hamlelerin kontrolünde ve Şah/Mat kontrolünde taraf kavramına ihtiyaç duyulmaktadır.

"GameBoard.fiftyMove" nesne özelliği ile elli hamle kuralı kontrolü yapılmaktadır. Elli hareket kuralı; elli tam hamle boyunca hiçbir piyon hamlesi olmazsa veya hiçbir taş hamlesi olmazsa oyunun berabere sonuçlanmasıdır. Klasik satrançta elli tam hamle hem beyazın elli hamlesi hem de siyahın elli hamlesi demektir. Yani toplamda yüz hamleden oluşmaktadır. Dolayısıyla "GameBoard.fiftyMove" değişkeni yüz değerine ulaştığında oyun berabere sonuçlanmaktadır. Ancak eş zamanlı satrançta taraflar aynı anda hareket ettiği için "GameBoard.fiftyMove" değişkeninin elli değerine ulaşmasıyla oyun berabere sonuçlanmaktadır.

"GameBoard.hisPly" nesne özelliği ile klasik satrançta kaçıncı yarım hamlenin yapıldığı bilgisini tutmaktadır. Ancak eş zamanlı satrançta hamleler aynı anda yapıldığı için bu değişkenle (nesneyle) kaçıncı eş zamanlı hamlenin yapıldığı bilgisi tutulmaktadır. Eş zamanlı satrancın doğası gereği yarım hamle bulunmamaktadır.

"GameBoard.enPas" dizininde, geçerken alma hamlesinin yapılacağı kare tutulmaktadır. Klasik satrançta bir pozisyonda sadece bir kare geçerken alma

karesi olabilirken eş zamanlı satrançta iki kare aynı anda geçerken alma karesi olabilmektedir. Bu nedenle iki elemanlı bir dizi tutulmaktadır.

"GameBoard.history" nesne özelliği bir dizindir. Bu dizin ile oyunun tüm pozisyonlarına ait bilgiler tutulmaktadır. Bunlar; oyuncunun hamlesi (playerMove), rakibin hamlesi (opponentMove), rok bilgisi (castlePerm), geçerken alma bilgisi (wEnPas ve bEnPas), elli hareket kuralı bilgisi (fiftyMove) ve pozisyon anahtarı (posKey) değerleridir. Bu dizin oyun esnasında hamlelerin geri ve ileri alınması için kullanılmaktadır.

"GameBoard.ply" nesne özelliği en iyi hamleyi bulmak için kullanılan arama ağacında kaçıncı yarım hamlede olunduğunu göstermektedir.

"GameBoard.castlePerm" nesne özelliği ile rok yapabilme bilgisi tutulmaktadır. Bu değer dört bitlik bir tamsayıdır. Her bir bit'in temsil ettiği değer aşağıdaki gibidir,

- 0001 : Beyaz Şah tarafı rok (White King Side Castling - WKCA)
- 0010 : Beyaz Vezir tarafı rok (White Queen Side Castling - WQCA)
- 0100 : Siyah Şah tarafı rok (Black King Side Castling - BKCA)
- 1000 : Siyah Vezir tarafı rok (White King Side Castling - BQCA)

Tarafların rok yapma yetkisinin olup olmadığı bilgisi "GameBoard.castlePerm" ile aşağıdaki CASTLEBIT değişkeninin bitsel VEYA (bitwise AND) işlemi sonucu belirlenmektedir. Örneğin; tüm bitler 1 ise yani, 1111 (onluk tabanda 15) bilgisi mevcutsa her iki taraf da hem uzun hem kısa rok yapabilmektedir. 0011 (on tabanında 3) bilgisi varsa sadece beyaz taraf hem uzun hem kısa rok yapabilmekte, siyah taraf ise rok yapamamaktadır. CASTLEBIT değişkeni defs.js dosyası içerisine aşağıdaki şekilde tanımlanmıştır,

```
var CASTLEBIT = { WKCA : 1, WQCA : 2, BKCA : 4, BQCA : 8 };
```

Örneğin beyaz tarafın şahın olduğu yöne doğru rok yetkisi olup olmadığını öğrenmek istediğimizde aşağıdaki kod parçasını çalıştırıyoruz;

```
"if (GameBoard.castlePerm & CASTLEBIT.WKCA != 0)"
```

Yukarıdaki kod **true** değeri dönerse beyaz tarafın kısa rok (şah tarafı) yetkisi olduğu anlaşılmaktadır.

"GameBoard.material" dizisiyle tarafların materyal değerleri tutulmaktadır. Bu değer en iyi hamlenin tarandığı arama ağacında pozisyonun değerlendirilmesi için kullanılmaktadır. Tahtadaki her bir taşın bir değeri vardır ve bu değerlerin toplamı her iki taraf için ayrı ayrı bu dizinde tutulmaktadır. Oyun taşlarının değerleri defs.js dosyası içerisinde aşağıdaki şekilde tanımlanmıştır,

var PieceVal= [0, 100, 325, 325, 550, 1000, 50000, 100, 325, 325, 550, 1000, 50000];

Bu değerler birinci indisten başlayarak sırasıyla, beyaz piyon, beyaz at, beyaz, beyaz fil, beyaz kale, beyaz vezir, beyaz şah, siyah piyon, siyah at, siyah fil, siyah kale, siyah vezir ve siyah şah taşlarının değerleridir.

Kodlamanın hızlı ve daha rahat bir şekilde yapılabilmesi için aşağıdaki diziler de "defs.js" dosyası içerisinde tanımlanmıştır. Bu dizilerde bir taşın hangi çeşit taş olduğu, hangi renk olduğu, büyük taş sınıfından mı yoksa küçük taş sınıfından mı olduğu gibi bilgiler tutulmaktadır. Bu dizilerin elemanları PIECES nesnesinin özellikleri (object property) ile aynı sıralamadadır. Örneğin; PIECES.wP (1. nesne özelliği) PieceBig dizisinin 1. indisiyle karşılaştırılmakta ve BOOL.FALSE değerine karşılık geldiği yani, büyük taş olmadığı görülmektedir. Diğer dizilerde de benzer bir mantık uygulanmıştır.

var PieceBig = [BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE];

"PieceBig" dizisi piyon ve diğer taşları ayırmak için kullanılmıştır. Bu dizi, bir taş büyük taşlardan mı sorgusuna piyon harici tüm taşlarda TRUE değeri dönmektedir.

var PieceMaj = [BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE];

"PieceMaj" dizisi sadece kale, vezir ve şah için TRUE değerleri tutmakta ve diğer taşlar için ise FALSE değerleri tutmaktadır.

var PieceMin = [BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE];

"PieceMin" dizisi küçük taş olarak değerlendirilen taşlar için (at, fil) TRUE değeri tutmakta ve diğer taşlar için ise FALSE değerleri tutmaktadır.

var PieceCol = [COLOURS.BOTH, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE, COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK];

"PieceCol" ile taşların hangi renkte olduğu bilgisi elde edilmektedir.

var PiecePawn = [BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE];

"PiecePawn" ile taşların piyon olup olmadığı bilgisi elde edilmektedir.

var PieceKnight = [BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE];

"PieceKnight" ile taşların at olup olmadığı bilgisi elde edilmektedir.

var PieceKing = [BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE];

" PieceKing" ile taşların şah olup olmadığı bilgisi elde edilmektedir.

var PieceRookQueen = [BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE];

"PieceRookQueen" ile taşların kale veya vezir olup olmadığı belirlenmektedir.

var PieceBishopQueen = [BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE];

"PieceBishopQueen" ile taşların fil veya vezir olup olmadığı belirlenmektedir.

var PieceSlides = [BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE];

"PieceSlides" ile taşların sırayla önündeki tüm kareler üzerinden kurallar çerçevesinde ilerleyip ilerleyemediği bilgisi elde edilmektedir. Örneğin; fil bu sınıfa girmekte ama at sadece "L" şeklinde hamle yapabildiği için bu sınıfa girememektedir.

3.4 Satır (Ranks) ve Sütun (Files) Dizileri

Satır ve sütun numaralarının tutulduğu iki farklı dizin tutulmaktadır. Bu dizinler kullanılarak oyun tahtasındaki bir karenin indisi elde edilmektedir.

```
var FilesBrd = new Array(BRD_SQ_NUM);
var RanksBrd = new Array(BRD_SQ_NUM);
```

Yukarıdaki dizin değerleri aşağıdaki **FR2SQ** fonksiyonuna parametre olarak verilmekte ve ilgili satır ve sütuna karşılık gelen kare indisi elde edilmektedir. Örneğin: 0. satır ve 0. sütundaki karenin indisi 21 olarak dönmektedir.

```
function FR2SQ(f,r) {
  return ( (21 + (f) ) + ( (r) * 10 ) );
}
```

Satır ve Sütun dizilerine değerlerinin atanma işlemi aşağıdaki **InitFilesRanksBrd** fonksiyonu ile yapılmaktadır. Bu fonksiyonda ilk for döngüsü (8. ve 11. satırlar arası) ile her iki dizine de OFFBOARD yani, tamsayı 100 değeri atanmaktadır. İkinci for döngüsünde (13. ve 19. satırlar arası) ise satır (RanksBrd) ve sütun (FilesBrd) dizilerine 100 haricindeki diğer satır ve sütun değerleri atanarak Şekil 3-1 ve Şekil 3-2'de görülen değerler dizilere atanmaktadır.

```
1  Function InitFilesRanksBrd() {
2
3      var index = 0;
4      var file = FILES.FILE_A;
5      var rank = RANKS.RANK_1;
6      var sq = SQUARES.A1;
7
8      for(index = 0; index < BRD_SQ_NUM; ++index) {
9          FilesBrd[index] = SQUARES.OFFBOARD;
10         RanksBrd[index] = SQUARES.OFFBOARD;
11     }
12
13     for(rank = RANKS.RANK_1; rank <= RANKS.RANK_8; ++rank) {
14         for(file = FILES.FILE_A; file <= FILES.FILE_H; ++file) {
15             sq = FR2SQ(file,rank);
16             FilesBrd[sq] = file;
17             RanksBrd[sq] = rank;
18         }
19     }
20 }
```

100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	0	0	0	0	0	0	0	0	100
100	1	1	1	1	1	1	1	1	100
100	2	2	2	2	2	2	2	2	100
100	3	3	3	3	3	3	3	3	100
100	4	4	4	4	4	4	4	4	100
100	5	5	5	5	5	5	5	5	100
100	6	6	6	6	6	6	6	6	100
100	7	7	7	7	7	7	7	7	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100

Şekil 3-1 Satır (Rank) Dizisi

100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	0	1	2	3	4	5	6	7	100
100	0	1	2	3	4	5	6	7	100
100	0	1	2	3	4	5	6	7	100
100	0	1	2	3	4	5	6	7	100
100	0	1	2	3	4	5	6	7	100
100	0	1	2	3	4	5	6	7	100
100	0	1	2	3	4	5	6	7	100
100	0	1	2	3	4	5	6	7	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100

Şekil 3-2 Sütun(File) Dizisi

3.5 Oyun Taşı Listeleri (Piece lists)

Satranç oyununda yapılan bir hamlenin doğru hamle olup olmadığını tespit etmek ve yapılabilecek hamleler içerisinden en iyi hamlenin belirlenmesi için olası hamlelerin tespit edilmesi gerekmektedir. Bu nedenle her bir taşın tahtada nerede olduğunun tespit edilmesinin ardından, yapabileceği hamlelerin listelenmesi gerekmektedir. Bir taş çeşidinin hangi karelerde olduğunun tespit edilebilmesi için Gameboard.pieces dizisinin tamamının bir döngü ile gezilmesi gerekmektedir. Bu da siyah ve beyaz taraf ayrı ayrı düşünüldüğünde 12 çeşit taş için 64 eleman uzunluğundaki bir dizinin 12 defa gezilmesi demektir. Bu yöntemle amaca ulaşılabilir fakat bu etkin olmayan bir yöntemdir. Bu durum sözde kod (Pseudo) ile aşağıdaki gibi gösterilebilir;

```
loop (pieces[])
  if(piece on sq == Side tomove)
  then genmoves() for piece on sq
end
```

Daha etkin bir arama ise hangi taştan ne kadar varsa sadece o uzunlukta döngüler kullanılarak yapılabilmektedir. Bunun için de programda iki ayrı dizi tutulmaktadır. Bu diziler board.js dosyası içerisinde aşağıdaki gibi tanımlanmıştır;

```
GameBoard.pceNum = new Array(13);
GameBoard.pList = new Array(13 * 10);
```

Hangi taştan kaç tane olduğunu tespit etmek için "GameBoard.pceNum" dizisi ve hangi çeşit taşın hangi karede olduğu bilgisine daha çabuk erişmek için ise "GameBoard.pList" dizisi tutulmaktadır. İlk bakışta bir taş çeşidinden en fazla iki tane var gibi görünse de taş terfilerini de düşündüğümüzde bir taş çeşidinden en fazla 10 tane olabilmektedir (vezir (en fazla 9 adet), şah (1 adet) ve piyon (8 adet) hariç). Bu kodlamada her bir taştan en fazla 10 tane olabileceği genellemesiyle hareket edilmiştir. "GameBoard.pList" tek boyutlu bir dizi olduğundan dolayı bir indeksleme geliştirilmiştir. Her biri 10 adet varsayılan 12 çeşit taşın hangi karelerde tutulduğu bilgisi taşı temsil eden numara (Örneğin: wP == 1) ile 10 tamsayısının çarpımı ve o taş çeşidinden kaçıncı taş olduğu sayısı eklenerek bulunur. Bu durum aşağıdaki kod ile daha iyi görülebilmektedir;

```
for(pceNum = 0; pceNum < GameBoard.pceNum[wP]; ++pceNum) {
  sq = GameBoard.pList [wP * 10 + pceNum]
  genmoves(sq) // ilgili kare için hamleler üretiliyor.
}
```

Bu durumda; wP: 10-19, wN: 20-29 ... bQ : 110-119 ve bK : 120-129 indeks numaralarında tutulmaktadır.

3.6 Pozisyon Özetleme Anahtarı (Position Hash Key)

Pozisyon özetleme anahtarı oyun esnasında üçlü tekrarlama olup olmadığının öğrenilmesi amacıyla kullanılmaktadır.

Pozisyonun eşsiz (unique) olup olmadığına yani, daha önce aynı pozisyonun tekrar edilip edilmediğini taşların tahta üzerindeki yerlerine, hamle sırasının hangi tarafta (Side) olduğuna (Sadece klasik satranç için geçerli. Eş zamanlı satrançta oyun sırası kavramı olmadığı için pozisyon özetleme anahtarında da böyle bir kavram bulunmamaktadır.), rok bilgisine (Castle) ve geçerken alma bilgisine (enPass) bakarak karar verilmektedir. Bunun yapılabilmesi için rastgele sayılar kullanılmaktadır. Rastgele sayı üretmek için kullanılan fonksiyon aşağıdaki şekli ile defs.js dosyası içerisinde yer almaktadır;

```
function RAND_32() {
  return (Math.floor((Math.random()*255)+1) << 23) |
  (Math.floor((Math.random()*255)+1) << 16) |
```

```
  (Math.floor((Math.random()*255)+1) << 8) |
  (Math.floor((Math.random()*255)+1));
}
```

Yukarıdaki fonksiyondaki **Math.random** fonksiyonuyla 0 ile 1 (hariç) arasında rastgele sayı üretilmektedir. Bu sayı 255 ile çarpılarak 0 ile 255 (hariç) arasında bir sayı elde edilmektedir. Bu sayıya da 1 eklenerek 0-256 (hariç) arası bir sayı elde edilmektedir. Son olarak da Math.floor() kullanılarak 0-255 arasında bir tam sayı elde edilmektedir. Başka bir deyişle; bu işlem ile ilk 8 biti bir değer içeren ve diğer bitleri 0 olan 32 bitlik bir tam sayı elde edilmektedir. Bu işlem dört defa yapılmakta ve soldan ilk üçü bitsel sola kaydırma işlemi ile sırasıyla 23, 16 ve 8 bit sola kaydırılmakta ardından da tüm değerlere bitsel VEYA (Bitwise OR) işlemi uygulanmaktadır. Sonuç olarak 31 bitlik pozitif bir tam sayı elde edilmektedir. Elde edilen bu değer, 23 bit sola kaydırılan rastgele sayı 24 bit sola kaydırıldığında 32 bit olabilirdi ancak kaydırma işlemi en soldaki biti 1 yaptığında sayı negatif olmaktadır. Bu durum rastgelelik açısından her ne kadar olumsuz bir durum oluşturmasa da kaynak kodlardaki bu yapı "eş zamanlı satranç" kodlamasında da korunmuştur. Bu kod parçası aşağıdaki gibi daha basit bir şekilde de yazılabilirdi. Böylelikle 32 bitlik rastgele sayılar elde edilebilirdi ancak burada da yine kaynak kod değiştirilmemiştir;

```
function RAND_32() {
  return (Math.floor((Math.random()*4294967295)+1);
}
```

"RAND_32()" fonksiyonu kullanılarak her bir taş çeşidine oyun tahtasındaki 64 kare için rastgele sayılar üretilmektedir. Aynı zamanda rok (Castle) ve geçerken alma karesi (enPass) bilgileri için de rastgele sayılar üretilmektedir. Eş zamanlı satrançta taraf (side) kavramı kalmadığı için yani, her iki taraf da aynı anda hareket ettiği için taraf (side) için bir rastgele sayı oluşturulmamaktadır. Her bir taşa ve diğer değerlere karşılık gelen rastgele sayıların hepsi birbiriyle bitsel XOR işlemine tabii tutularak mevcut pozisyona ait pozisyon anahtar değeri elde edilmektedir. Bu işlem oyun esnasında her eş zamanlı hamleden sonra yapılmaktadır. Eğer aynı değer üç defa elde edilirse üçlü tekrarlama kuralından dolayı oyun berabere sonuçlandırılmaktadır.

Yukarıda anlatılan "pozisyon özetleme anahtarı" ile ilgili kodlama için aşağıdaki değişkenler defs.js dosyasında tanımlanmıştır;

```
var PieceKeys = new Array(13 * 120);
var CastleKeys = new Array(16);
```

"PieceKeys" dizisinde her bir taşın 120 farklı karede alacağı farklı rastgele sayılar tutulmaktadır. Ayrıca bu dizinin 0-119 numaralı indis aralığında ise enPass karesi oluştuğunda kullanılmak üzere rastgele sayılar tutulmaktadır. Bu dizi 64 eleman uzunluğunda da olabilirdi ancak hem kodlamada kolaylık olmasından hem de oyun başlangıcında bir defaya mahsus olarak çalıştırıldığından dolayı performansı etkilemeyeceğinden bu şekilde kodlanmıştır.

"CastleKeys" dizisinde ise rok bilgilerine ait rastgele sayılar tutulmaktadır. Rok bilgisi 4 bit uzunluğunda bir değer ile temsil edildiğinden dolayı 16 eleman uzunluğunda bir dizi kullanılmaktadır.

Yukarıda açıklaması yapılan rastgele sayıların üretilmesi ve dizilere kaydedilmesi ile ilgili kodlama main.js dosyasında aşağıdaki şekilde yer almaktadır;

```
1  function InitHashKeys() {
2      var index = 0;
3      for(index = 0; index < 13 * 120; ++index) {
4          PieceKeys[index] = RAND_32();
5      }
6      for(index = 0; index < 16; ++index) {
7          CastleKeys[index] = RAND_32();
8      }
9  }
```

Şu ana kadar "pozisyon özetleme anahtarı" için gerekli ön işlemler yapılmış bulunmaktadır. Aşağıda ise board.js dosyası içerisinde bulunan "pozisyon özet anahtarının" oluşturulduğu **GeneratePosKey** fonksiyonu yer almaktadır. Bu fonksiyondaki for döngüsünde oyun tahtasındaki her bir kare gezilmektedir (5. ve 10. satırlar arası). Şayet gezilen oyun karesinde bir taş varsa ve kare SQUARES.OFFBOARD değilse yani, kare asıl oyun tahtasını oluşturan 64 kareden biriyse o karede bulunan taşın rastgele sayısı ile finalKey değişkeni XOR işlemine tabii tutulmaktadır. Ayrıca oyunculardan herhangi biri (veya her ikisi) tarafından geçerken alma karesi (enPass) oluşturulmuşsa o kare(lere) ait rastgele sayı(lar) da finalKey değişkeni ile XOR işlemine tabii tutulmaktadır (11. ve 16. satırlar arası). Son olarak, rok bilgisi için tutulan rastgele sayı ile finalKey XOR

işlemine tabii tutularak pozisyon özetleme anahtarı oluşturulmaktadır (17. satır). 18. satırda da oluşturulan pozisyon özetleme anahtarı döndürülmektedir.

```
 1  function GeneratePosKey() {
 2      var sq = 0;
 3      var finalKey = 0;
 4      var piece = PIECES.EMPTY;
 5      for(sq = 0; sq < BRD_SQ_NUM; ++sq) {
 6          piece = GameBoard.pieces[sq];
 7          if(piece != PIECES.EMPTY && piece != SQUARES.OFFBOARD) {
 8              finalKey ^= PieceKeys[(piece * 120) + sq];
 9          }
10      }
11      if(GameBoard.enPas[0] != SQUARES.NO_SQ) {
12          finalKey ^= PieceKeys[GameBoard.enPas[0]];
13      }
14      if(GameBoard.enPas[1] != SQUARES.NO_SQ) {
15          finalKey ^= PieceKeys[GameBoard.enPas[1]];
16      }
17      finalKey ^= CastleKeys[GameBoard.castlePerm];
18      return finalKey;
19  }
```

3.7 64 ve 120 Eleman Uzunluğundaki Dizi Dönüşümleri

Bilindiği üzere satranç tahtası 64 kareden oluşmaktadır ama şu ana kadar yapılan kodlamada 120 kareyi temsil eden 120 eleman uzunluğundaki dizi kullanılmıştır. Bu durumun şu ana kadar performansa önemli derecede olumsuz etkisi olmamıştır. Ancak bundan sonraki kodlamalarda bu durum performansı etkilemektedir. Bu nedenle 120 yerine 64 uzunluğundaki dizi kullanılarak etkinlik arttırılmaktadır. Bunun için de 64 ve 120 uzunluğundaki dizilerde birbirinin karşılığı olan indis numaralarının bulunması için gerekli dönüşümlerin yapılması gerekmektedir. Bu dönüşümler için kullanılacak olan dizi tanımlamaları defs.js içerisinde aşağıdaki şekilde yapılmıştır;

```
var Sq120ToSq64 = new Array(BRD_SQ_NUM);
var Sq64ToSq120 = new Array(64);
```

Bu dönüşüm fonksiyonu ile 64 eleman uzunluğundaki dizideki bir karenin 120 eleman uzunluğundaki dizideki karşılığı ve tersi işlemi yapılarak dizi

dönüşümleri sağlanmaktadır. Bu işlemler için main.js dosyası içerisinde yer alan aşağıdaki **InitSq120To64** fonksiyonu kullanılmaktadır;

```
 1  function InitSq120To64() {
 2      var index = 0;
 3      var file = FILES.FILE_A;
 4      var rank = RANKS.RANK_1;
 5      var sq = SQUARES.A1;
 6      var sq64 = 0;
 7      for(index = 0; index < BRD_SQ_NUM; ++index) {
 8          Sq120ToSq64[index] = 65;
 9      }
10      for(index = 0; index < 64; ++index) {
11          Sq64ToSq120[index] = 120;
12      }
13      for(rank = RANKS.RANK_1; rank <= RANKS.RANK_8; ++rank) {
14          for(file = FILES.FILE_A; file <= FILES.FILE_H; ++file) {
15              sq = FR2SQ(file,rank);
16              Sq64ToSq120[sq64] = sq;
17              Sq120ToSq64[sq] = sq64;
18              sq64++;
19          }
20      }
21  }
```

InitSq120To64 fonksiyonunun 7. ve 9. satırları arasında, Sq120ToSq64 dizisinde tutulmayan bir değer olan 65 sayısı tüm dizi elemanlarına atanarak Sq120ToSq64 dizisi ilklenmektedir. 10. ve 12. satırlar arasında ise benzer işlem "Sq64ToSq120" dizisinin elemanlarına 120 değeri atanarak yapılmaktadır. 13. ve 20. satırlar arasında ise içi içe iki döngü kullanılarak bu dizilerin tutması gereken değerler atanmaktadır. Yani, "Sq120ToSq64" dizisinin 120'den 64'e ve "Sq64ToSq120" dizisinin de 64'ten 120'ye dönüşüm sağlaması için gerekli olan değerler atanmaktadır. Tahtadaki karelerin üzerinde dolaşmak için de satır ve sütun değerleri kullanılmaktadır. İç içe döngüler ile sırasıyla ilk satırın tüm sütunları gezilmektedir, sonrasında da sırayla tüm satır ve sütunlar benzer şekilde gezilmektedir. Aynı zamanda bu satır ve sütun değerleri kullanılarak (15. satır) 120 eleman uzunluğundaki tahtanın oyun tahtası dışı olmayan ilk karesi olan 21. kareden başlayan karelerin indis numarası elde edilmektedir. 16. ve 17. satırlarda, dizilerin dönüşüm için tutması gereken değerleri atanmaktadır. Örneğin;

"Sq64ToSq120[0] = 21;" ataması yapılmaktadır. 64 eleman uzunluğundaki diziden 120 eleman uzunluğundaki diziye dönüşüm ihtiyacı duyulduğunda 64 elemanlı dizinin "0." indisinin döndüreceği değer "21" olacaktır. Bu değer de 120 eleman uzunluğundaki dizinin ilk aktif oyun tahtası karesi olan 21. indisi vermektedir. Bu döngüde "Sq120ToSq64" dizisine 21. indisten itibaren atama yapılmaktadır. Çünkü öncesindeki alanların hepsi oyun tahtasının dışı olarak tanımlanmaktadır.

3.8 FEN Pozisyon Metni (FEN Position String)

Forsyth-Edward Notation (FEN) metni, belirli bir satranç tahtası pozisyonunu gösteren bir notasyondur. Oyuna kalındığı yerden tekrar başlamak için gerekli tüm bilgiler FEN ile kaydedilebilmektedir. FEN metnini oyuncunun kendisi de oluşturabilmekte ve bu metni oyun tahtasına transfer ederek oyuna o pozisyondan başlayabilmektedir. FEN metni altı bölümden oluşmaktadır. Aşağıda standart açılış pozisyonu için kullanılan bir FEN metni görülmektedir;

"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - - 0 0"

Turkuaz renkle işaretlenmiş olan ilk kısımda tahtadaki taşlar ve boş alanlar gösterilmektedir. Küçük harfler siyah tarafın büyük harfler ise beyaz tarafın taşlarını temsil etmektedir. "r/R" Rook (kale), "n/N" kNight (at), "b/B" Bishop (fil), "q/Q" Queen (vezir), "k/K" King (Şah) ve "p/P" Pawn (piyon)'a karşılık gelmektedir. "/" bir sonraki satıra geçilmesi gerektiğini belirtmektedir. Rakamlar (bu örnekte 8) boş kareleri temsil etmektedir. Pembe renk ile işaretlenmiş olan ikinci kısım hamle sırasının hangi tarafta olduğunu göstermektedir. Eş zamanlı satranç oyununda taraflar aynı anda hamle yaptıkları için bunun bir önemi kalmamıştır. Açık gri renkle işaretlenmiş olan üçüncü kısımda rok yetkisi belirtilmektedir. "K" beyaz tarafın şah tarafına doğru (kısa rok) rok yapma yetkisi olduğunu, "Q" yine beyaz tarafın vezir tarafına doğru (uzun rok) rok yapma yetkisi olduğunu göstermektedir. "k" ve "q" da siyah taraf için sırasıyla kısa rok ve uzun rok yapma yetkilerini göstermektedir. Şayet hiç rok yetkisi yoksa bu durum "-" ile belirtilmektedir. Ayrıca ne kadar rok yetkisi varsa ona karşılık gelen harfler gösterilmektedir. Örneğin; "Kq" beyaz kısa rok ve siyah uzun rok yetkilerinin olduğunu göstermektedir. Sarı ile işaretlenmiş olan dördüncü kısımda enPass (geçerken alma) karesi bilgisi yer almaktadır. Klasik satrançtan farklı olarak eş zamanlı satrançta bir değil iki adet enPass karesi oluşabildiğinden dolayı bu alanda iki taraf içinde enPass karesi bilgisi yer almaktadır. "- -" her iki taraf için de enPass karesi olmadığını göstermektedir. enPass karesi varsa da o karenin

bilgisi alfa-nümerik olarak yer almaktadır. Örneğin:"e6 -" beyaz tarafta e6 karesinin enPass karesi olduğu ve siyah tarafta herhangi bir enPass karesi olmadığını göstermektedir. Yeşil ile işaretli beşinci kısımda elli hamle kuralı bilgisi yer almaktadır. Koyu gri ile işaretli altıncı ve son kısımda ise yarım hamle sayısı yer almaktadır. Bu alan eş zamanlı satrançta tam hamle sayısını ifade etmektedir. Bu son iki alan bu uygulamada dikkate alınmamaktadır.

FEN metninin kullanıcıdan alınarak işlenmesinden önce oyun tahtasının ilklenmesi gerekmektedir. Bu nedenle oyun tahtasının ilklenmesi ile ilgili olan ve programın ilerleyen aşamalarında da kullanılacak olan bazı değişkenler defs.js dosyası içerisine aşağıdaki şekilde tanımlanmaktadır;

```
var MAXGAMEMOVES = 2048;
var MAXPOSITIONMOVES = 256;
var MAXDEPTH = 64;
```

Bir oyun boyunca yapılacak maksimum hamle sayısı MAXGAMEMOVES değişkeni ile 2048 olarak belirlenmektedir. Bir oyun pozisyonunda üretilebilecek maksimum hamle sayısı MAXPOSITIONMOVES değişkeni ile belirlenmektedir. Yani 256'dır. Bu sayı da bir pozisyonda bir taraf için oldukça yeterlidir. Oyun esnasında bilgisayarın en iyi hamlesini belirleyebilmesi için mevcut pozisyondan sonraki pozisyonları da değerlendirmesi gerekmektedir. Bu nedenle mevcut pozisyondaki hamleleri üretip bir diziye kaydettikten sonra sonraki pozisyonlardaki hamleleri de üretmesi ve bir diziye kaydetmesi gerekmektedir. Bilgisayarın art arda yapacağı pozisyon değerlendirmelerine derinlik adı verilmektedir ve bu uygulamada derinlik MAXDEPTH değişkeni kullanılarak 64 olarak belirlenmiştir. MAXPOSITIONMOVES ve MAXDEPTH değerlerinin çarpımı hamlelerin kaydedileceği dizinin boyutunu vermektedir.

Mevcut oyun pozisyonunda yapılabilecek hamleler ve o hamlelerle ilgili değerlerin kaydedilmesi için aşağıdaki tanımlamalar board.js dosyası içerisinde yer almaktadır;

```
GameBoard.moveList = new Array(MAXDEPTH *
                               MAXPOSITIONMOVES);
GameBoard.moveScores = new Array(MAXDEPTH *
                               MAXPOSITIONMOVES);
GameBoard.moveListStart = new Array(MAXDEPTH);
```

GameBoard.moveList dizisine her bir derinlikte yapılabilecek hamleler kaydedilmektedir. GameBoard.moveScores dizisine ise bu hamlelerin değerleri kaydedilmektedir. GameBoard.moveListStart dizisi ile de GameBoard.moveList'in belirli bir derinlikteki başlangıç indeksi tutulmaktadır.

Oyun tahtasının sıfırlanması ile ilgili ön gereklilikler yukarıda tanımlandıktan sonra sıfırlama işleminin yapıldığı **ResetBoard** fonksiyonu board.js dosyası içerisine aşağıdaki şekilde yer almaktadır;

```
1  function ResetBoard() {
2      var index = 0;
3      for(index = 0; index < BRD_SQ_NUM; ++index) {
4          GameBoard.pieces[index] = SQUARES.OFFBOARD;
5      }
6      for(index = 0; index < 64; ++index) {
7          GameBoard.pieces[SQ120(index)] = PIECES.EMPTY;
8      }
9      GameBoard.side = COLOURS.BOTH;
10     GameBoard.enPas[0] = SQUARES.NO_SQ;
11     GameBoard.enPas[1] = SQUARES.NO_SQ;
12     GameBoard.fiftyMove = 0;
13     GameBoard.ply = 0;
14     GameBoard.hisPly = 0;
15     GameBoard.castlePerm = 0;
16     GameBoard.posKey = 0;
17     GameBoard.moveListStart[GameBoard.ply] = 0;
18 }
```

ResetBoard fonksiyonunda 3. ve 5. satırlar arasındaki döngüde tüm oyun tahtası SQUARES.OFFBOARD değeriyle doldurulmaktadır. 6. ve 8. satırlar arasında ise asıl oyun tahtasını oluşturan 64 kare PIECES.EMPTY değeriyle doldurulmaktadır. Böylelikle boş bir oyun tahtası elde edilmiş olunmaktadır. 9. ve 17. satırlar arasında, GameBoard nesnesinin diğer özellilerine (değişkenler) de oyun başlangıç durumundaki değerler atanmaktadır. "GameBoard.side = COLOURS.BOTH;" ifadesi hamle sırasının her iki tarafta da olmadığını göstermektedir. "GameBoard.enPas[0] = SQUARES.NO_SQ;" ve "GameBoard.enPas[1] = SQUARES.NO_SQ;" ifadeleri de her iki tarafında geçerken alma karelerinin sıfırlandığını göstermektedir. " GameBoard.moveListStart[GameBoard.ply] = 0;" ifadesi ise her bir derinlikteki

hamlelerin listelendiği GameBoard.moveList dizisi üzerinde hangi derinlikte hangi indeksten başlanacağı bilgisinin sıfırlandığını göstermektedir.

Bir FEN metnin çözümlenerek oyun tahtası üzerinde yerleştirilmesi için board.js dosyası içerisinde yer alan, aşağıdaki **ParseFen** fonksiyonu kullanılmaktadır;

```
function ParseFen(fen) {
  ResetBoard();
  var rank = RANKS.RANK_8;
  var file = FILES.FILE_A;
  var piece = 0;
  var count = 0;
  var i = 0;
  var sq120 = 0;
  var fenCnt = 0;
  while ((rank >= RANKS.RANK_1) && fenCnt < fen.length) {
    count = 1;
    switch (fen[fenCnt]) {
      case 'p': piece = PIECES.bP; break;
      case 'r': piece = PIECES.bR; break;
      case 'n': piece = PIECES.bN; break;
      case 'b': piece = PIECES.bB; break;
      case 'k': piece = PIECES.bK; break;
      case 'q': piece = PIECES.bQ; break;
      case 'P': piece = PIECES.wP; break;
      case 'R': piece = PIECES.wR; break;
      case 'N': piece = PIECES.wN; break;
      case 'B': piece = PIECES.wB; break;
      case 'K': piece = PIECES.wK; break;
      case 'Q': piece = PIECES.wQ; break;
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
      case '8':
        piece = PIECES.EMPTY;
        count = fen[fenCnt].charCodeAt() - '0'.charCodeAt();
        break;
```

```
36          case '/':
37          case ' ':
38              rank--;
39              file = FILES.FILE_A;
40              fenCnt++;
41              continue;
42          default:
43              console.log("FEN error");
44              return;
45      }
46      for (i = 0; i < count; i++) {
47          sq120 = FR2SQ(file,rank);
48          GameBoard.pieces[sq120] = piece;
49          file++;
50      }
51      fenCnt++;
52  } // while loop end
53  GameBoard.side = (fen[fenCnt] == 'w') ? COLOURS.WHITE :
    COLOURS.BLACK;
54  fenCnt += 2;
55  if ( (fen[fenCnt] == 'K') || (fen[fenCnt] == 'Q') || (fen[fenCnt] == 'k') ||
    (fen[fenCnt] == 'q') ){
56      for (i = 0; i < 4; i++) {
57          if (fen[fenCnt] == ' ') {
58              break;
59          }
60          switch(fen[fenCnt]) {
61              case 'K': GameBoard.castlePerm |= CASTLEBIT.WKCA; break;
62              case 'Q': GameBoard.castlePerm |= CASTLEBIT.WQCA; break;
63              case 'k': GameBoard.castlePerm |= CASTLEBIT.BKCA; break;
64              case 'q': GameBoard.castlePerm |= CASTLEBIT.BQCA; break;
65              default:
                                break;
66          }
67          fenCnt++;
68      }
69      fenCnt++;
70  }
71  if (fen[fenCnt] != '-') {
72    file = fen[fenCnt].charCodeAt() - 'a'.charCodeAt();
73    rank = fen[fenCnt + 1].charCodeAt() - '1'.charCodeAt();
74    console.log("fen[fenCnt]:" + fen[fenCnt] + " File:" + file+"Rank:"+rank);
75    GameBoard.enPas[0] = FR2SQ(file,rank);
76    fenCnt +=3;
```

```
77    }else {fenCnt +=2;}
78    if (fen[fenCnt] != '-') {
79        file = fen[fenCnt].charCodeAt() - 'a'.charCodeAt();
80        rank = fen[fenCnt + 1].charCodeAt() - '1'.charCodeAt();
81        console.log("fen[fenCnt]:" + fen[fenCnt] + " File:"+file+" Rank:"+rank);
82        GameBoard.enPas[1] = FR2SQ(file,rank);
83    }
84    GameBoard.posKey = GeneratePosKey();
85    UpdateListsMaterial();
86  }//End of ParseFen Function
```

Yukarıda kodları verilen **ParseFen** fonksiyonunda öncelikle daha önce tanımlanmış olan **ResetBoard** fonksiyonu çağrılarak (2. satır) oyun tahtası sıfırlanmaktadır. Ardından fonksiyonda kullanılacak olan değişkenler tanımlanmaktadır (3. ve 9. satırlar arasında). "rank" değişkenine çözümleme esnasında başlangıç satırı olarak kullanılacak olan RANKS.RANK_8 değeri, yani sekizinci satır değeri atanmaktadır. FEN metnindeki veriler oyun tahtasının sekizinci satırından başlanarak aşağı doğru yerleştirilmektedir. Bir while döngüsü ile RANKS.RANK_1 (dahil) değerine ulaşılana kadar tarama yapılmaktadır. "file" değişkenine ise başlangıç olarak kullanılacak olan FILES.FILE_A değeri, yani A satırı (rakamsal olarak 0) atanmaktadır. Her bir "/" değeriyle karşılaşıldığında "file" değişkeninin değeri bir arttırılmaktadır. FEN metni içerisinde bir boşluk değeri ile karşılaşıldığında ise while döngüsünden çıkılmaktadır. "piece" değişkenine boş anlamına gelen "0" değeri atanmıştır. fenCnt değişkeni FEN metin dizisindeki karakterlerin indeksi olarak kullanılmaktadır. "0" değeri atanarak dizideki ilk karakterden tarama yapılmaya başlanmaktadır. Bir karakter taranıp işleme alındıktan sonra fenCnt değeri bir arttırılmaktadır. "count" değişkenine FEN metni içerisinde bir tarama yapılırken 1 değeri atanmaktadır. Boşlukları temsil eden bir rakam (Örneğin; 8 - tüm satırın boş olduğunu göstermektedir.) taranırken de o rakam atanmaktadır. Bu atanan değer kadar döngü sürdürülmekte ve ilgili karelere boş kare değeri (PIECES.EMPTY) atanmaktadır. 10. satır ile 54. satır arasında bir while döngüsü kullanılarak FEN metni ile verilen oyun taşlarının neler olduğu ve nerelere yerleştirileceği belirlenmekte ve taşlar belirlenen yerlere yerleştirilmektedir. While döngüsü, "rank" değeri RANKS.RANK_1 (0)'e büyük eşit olduğu sürece ve FEN metnin sonuna ulaşana kadar çalıştırılmaktadır. 11. satırda count değişkenine 1 değeri atanmaktadır. FEN metni içerisinde taranan karakter bir oyun taşı ise count değeri 1 olarak kalmaktadır. Şayet boş kareleri temsil eden 1 - 8 arası bir değer ise o değer count değişkenine atanmaktadır. Oyun tahtasına

(GameBoard.pieces[sq120]) atanan bu değer kadar boş kare yerleştirilmektedir (46. ve 50. satırlar arası). 12. satırda FEN metninin parametre olarak verildiği bir switch case yapısı kullanılmaya başlanmaktadır. Bu yapı ile taranan karakter tespit edilmekte ve gerekli işlem yapılmaktadır. Rastlanılan karakterin bir oyun taşını temsil etmesi durumunda kodlamada o taşın temsil edildiği değer "piece" değişkenine atanmaktadır (13 ve 24. satırlar arası). Taranan değerin rakam olup olmadığı 25 ve 32. satırlar arasında kontrol edilmektedir. Şayet rakam ile karşılaşılırsa piece değişkenine 33. satırda boş kare değeri atanmaktadır. 34. satırda ise count değişkenine kaç tane boş kare olacağı, yani karşılaşılan rakamın değeri integer'a dönüştürülerek atanmaktadır. 35. satırda ise switch case içerisinden çıkılmaktadır. 36. veya 37. satırlardaki şartlar sağlandığında yani, sıradaki karakterin '/' veya ' ' (boşluk karakteri) olma durumunda ise rank değişkeninin değeri bir azaltılarak bir alt satıra geçilmekte (38. satır), ilk sütuna dönülmesi için gerekli olan değer (FILES.FILE_A) file değişkenine atanmakta (40. satırda) ve karakter taramasına devam edilmesi için switch case yapısının başlangıç noktasına dönülerek (42. satırda) taramaya bir sonraki karakterden devam edilmektedir. switch case yapısında şartı hiçbir case'e uymayan karakter ile karşılaşıldığında ise 42. satırdaki default case altındaki 43. ve 44. satırlar çalıştırılmaktadır. 43. satırda konsola bir hata mesajı yazdırılmaktadır. 44. satırda ise **ParseFen** fonksiyonundan çıkılmaktadır. 46. ve 50. satırlar arasındaki for döngüsü kullanılarak FEN metin içerisinde tespit edilen oyun taşı veya oyun taşı olmayan boş kareler oyun tahtasına (GameBoard.pieces[sq120]) yerleştirilmektedir. 51. satırda, fenCnt değeri bir arttırılarak bir sonraki FEN metin karakterinin taranması sağlanmaktadır. 52. satırda while döngüsünün sonlanmaktadır. Döngü şartları sağlanıyorsa döngüye devam edilmekte, sağlanmıyorsa fonksiyon bir sonraki satırdan çalışmaya devam etmektedir. 53. satırda FEN metni içerisinde karşılaşılması beklenen karakter 'w' veya 'b' dir. Bu karakterler oyun sırasının kimde olduğunu göstermektedir. Karşılaşılan karakter 'w' ise COLOURS.WHITE, 'b' ise COLOURS.BLACK değeri Gameboard.side değişkenine atanmaktadır. Bu değer eş zamanlı satranç oyununda kullanılmamaktadır. Bir sonraki FEN metin karakterinin taranması için fenCnt değeri 54. satırda iki arttırılmaktadır. 55. satırda, sıradaki karakterin 'K', 'Q', 'k' veya 'q' olup olmadığı sorgulanmaktadır. Bu karakterlerin hepsi veya bir kısmı FEN metni içerisinde bulunabilmekte veya herhangi biri bulunmayabilmektedir. Bu dört karakter için 56. ve 68. satırlar arasında bir for döngüsü çalıştırılmaktadır. FEN metninde taranan alanda " " (boşluk karakteri) ile karşılaşıldığında for döngüsünden çıkılmaktadır (57. ile 59. satırlar). 60. ve 66. satırlar arasında FEN metnindeki karaktere denk gelen rok yetkisi bir switch case yapısı ile

GameBoard.castlePerm değişkenine atanmaktadır. 67. satırda fenCnt değeri bir attırılmakta ve bir sonraki karakter analiz edilmektedir. FEN metninde rok karakterlerinden sonra gelen karakterleri analiz etmek için 69. satırda fenCnt değeri bir arttırılmaktadır. 71. ve 76. satırlar arasında beyaz taraf için enPass (geçerken alma) karesi olup olmadığı kontrol edilmekte, varsa da ilgili değişkene atanmaktadır. 71. satırda kontrol edilen karakter '-' ise enPass karesi yok emektir. Bu durumda 77. satırdaki kod çalıştırılarak fenCnt iki ilerletilerek devam edilmektedir. Şayet bir enPass karesi varsa bu kare iki karakter ile gösterilmektedir (c5 gibi). İlk karakter sütun değerini ikinci karakter ise satır değerini göstermektedir. 72. satırda, geçerken alma karesinin sütun değeri file değişkenine integer değere çevrilerek atanmaktadır. 73. satırda da benzer şekilde satır değeri rank değişkenine atanmaktadır. 78. ve 83. satılar arasında aynı işlemler siyah taraf için yapılmaktadır. 84. satırda **GeneratePosKey** fonksiyonu ile elde edilen pozisyon anahtarı değeri GameBoard.posKey değişkenine atanmaktadır. Daha önce bahsedildiği gibi bu değer üçlü tekrar olup olmadığı kontrolünde kullanılmaktadır. 85. satırda oyun tahtasındaki materyal bilgisi güncellenmektedir. 86. satırda da fonksiyon sonlanmaktadır.

3.9 Materyal Listesinin Güncellenmesi

Materyal listesinin(hangi taştan ne kadar olduğu bilgisi) güncellenmesi için **UpdateListsMaterial** fonksiyonu kullanılmaktadır. Bu fonksiyon içerisinde kullanılan Gameboard.pList isimli bir dizi bulunmaktadır. Bu dizi kullanılarak oyun tahtasındaki bütün kareleri gezinmeden bir oyun taşı tipine direk erişim sağlanabilmektedir. Gameboard.pList dizisinde, her bir taş çeşidinin hangi karede olduğu bir indeksleme yöntemi kullanılarak kaydedilmektedir. Bu indeksleme aşağıdaki gibidir;

piece * 10 + pieceIndexNumberInLoop

piece, her bir taşın uygulama içerisindeki temsil edildiği numarayı temsil etmektedir. Örneğin; Beyaz piyon (PIECES.wP) "1" rakamı ile temsil edilmektedir. Oyun başlangıcında da 8 tane piyon bulunmaktadır. Bu durumda indeksleme;

1*10 + 0 = 10
1*10 + 1 = 11
1*10 + 2 = 12

1*10 + 3 = 13..... şeklinde olmaktadır. Belirlenen indekslere ait dizideki alanlara da o tipteki oyun taşlarının hangi karelerde olduğu kaydedilmektedir. Pusedo kodu da aşağıdaki şekildedir;

```
pieceNumber[wP] = 8;
for (num = 0 to 7){
        wP * 10 + num; //10,11,12...
        sq = pList[10].....
}
```

Bir oyun taşına ait indekslemenin hesaplanması defs.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
function PCEINDEX(pce, pceNum) {
        return (pce * 10 + pceNum);
}
```

UpdateListsMaterial fonksiyonu board.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
 1  function UpdateListsMaterial() {
 2    var piece,sq,index,colour;
 3    for(index = 0; index < 13 * 10; ++index) {
 4      GameBoard.pList[index] = PIECES.EMPTY;
 5    }
 6    for(index = 0; index < 2; ++index) {
 7      GameBoard.material[index] = 0;
 8    }
 9    for(index = 0; index < 13; ++index) {
10      GameBoard.pceNum[index] = 0;
11    }
12    for(index = 0; index < 64; ++index) {
13      sq = SQ120(index);
14      piece = GameBoard.pieces[sq];
15      if(piece != PIECES.EMPTY) {
16        colour = PieceCol[piece];
17        GameBoard.material[colour] += PieceVal[piece];
18        GameBoard.pList[PCEINDEX(piece,GameBoard.pceNum[piece])]
          = sq;
19        GameBoard.pceNum[piece]++;
20      }
21    }
```

22 }

UpdateListsMaterial fonksiyonunun 3. ve 5. satırları arasında GameBoard.pList dizisinin içeriği sıfırlanmaktadır. 6. ve 8. satırlar arasında hangi tarafın oyun taşı değerinin ne kadar olduğunu tutan GameBoard.material dizisinin içeriği sıfırlanmaktadır. 9. ve 11. satırlar arasında hangi oyun taşından kaç tane olduğu bilgisini tutan GameBoard.pceNum dizisinin içeriği sıfırlanmaktadır. 12. satırda 64 indeks uzunluğunda bir for döngüsü başlatılmaktadır. 13. satırda mevcut indeks numarasının 120 eleman uzunluğundaki oyun tahtasındaki karşılığı belirlenmektedir. 14. satırda 120 elemanlı oyun tahtasının ilk karesindeki (21 numaralı kare) değer piece değişkenine atanmaktadır. Bu değer boş kare değilse if yapısının içerisine girilmektedir (15. satır). 16. satırda ilgili oyun taşının rengi colour değişkenine atanmaktadır. 17. satırda o oyun taşının taşıdığı değer GameBoard.material dizisinde kendi renginin değerinin tutulduğu alana eklenmektedir. 18. satırda oyun taşının bulunduğu kare bilgisi GameBoard.pList dizisinde **PCEINDEX(piece, GameBoard.pceNum[piece])** fonksiyonu ile hesaplanan indeks numarasının gösterdiği alana kaydedilmektedir. 19. satırda oyun taşı adetlerinin bilgisinin tutulduğu GameBoard.pceNum dizisindeki ilgili hane bir arttırılmaktadır.

3.10 Oyun Tahtasının Konsola Yazdırılması

Debug işlemlerinin yapılabilmesi için oyun tahtası konsola yazdırılmaktadır. Oyun tahtasının konsola yazdırılması için bazı tanımlamaların yapılması gerekmektedir. Bunlardan birisi olan standart satranç açılışında kullanılan FEN metni tanımlaması defs.js dosyasında aşağıdaki şekilde tanımlanmaktadır;

```
var START_FEN = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR
b KQkq - -";
```

Ayrıca oyun tahtası yazdırılırken kolaylık sağlaması için defs.js dosyası içerisinde tanımlı olan aşağıdaki karakter dizileri de kullanılmaktadır;

```
var PceChar = ".PNBRQKpnbrqk";
var SideChar = "wb-";
var RankChar = "12345678";
var FileChar = "abcdefgh";
```

Oyun tahtasının konsola yazdırılması için board.js içerisinde yer alan aşağıdaki **PrintBoard** fonksiyonu kullanılmaktadır;

```
1  function PrintBoard() {
2      var sq,file,rank,piece;
3      console.log("\nGame Board:\n");
4      for(rank = RANKS.RANK_8; rank >= RANKS.RANK_1; rank--) {
5          var line =(RankChar[rank] + "  ");
6          for(file = FILES.FILE_A; file <= FILES.FILE_H; file++) {
7              sq = FR2SQ(file,rank);
8              piece = GameBoard.pieces[sq];
9              line += (" " + PceChar[piece] + " ");
10         }
11         console.log(line);
12     }
13     console.log("");
14     var line = "    ";
15     for(file = FILES.FILE_A; file <= FILES.FILE_H; file++) {
16                 line += (' ' + FileChar[file] + ' ');
17     }
18     console.log(line);
19     console.log("enPas:"+GameBoard.enPas[0]+" - "+GameBoard.enPas[1]);
20     line = "";
21     if(GameBoard.castlePerm & CASTLEBIT.WKCA) line += 'K';
22     if(GameBoard.castlePerm & CASTLEBIT.WQCA) line += 'Q';
23     if(GameBoard.castlePerm & CASTLEBIT.BKCA) line += 'k';
24     if(GameBoard.castlePerm & CASTLEBIT.BQCA) line += 'q';
25     console.log("castle:" + line);
26     console.log("key:" + GameBoard.posKey.toString(16));
27 }
```

PrintBoard fonksiyonunun 2. satırında değişken tanımlamaları yapılmaktadır. 3. satırda "Game Board" şeklinde başlık niteliğinde bir metin konsola yazdırılmaktadır. 4. satırda oyun tahtasının sekizinci satırından başlayıp birinci satırına kadar giden bir for döngüsü başlatılmaktadır. 5. satırda line isminde bir değişken oluşturulmaktadır. Bu line değişkenine sırasıyla her bir satırda bulunan metin bilgisi eklenecektir. İlk olarak ekrana yazdırılan satır olan sekizinci satırın bilgisi (8) ve bir boşluk karakteri line değişkenine atanmaktadır. 6. satırda bir for döngüsü daha başlatılmaktadır. Bu for döngüsünde ise ilgili satırın sütunları line değişkenine eklenmektedir. 7. satırda analiz edilen satır (rank) ve sütun (file) bilgisi kullanılarak oyun karesi bilgisi elde edilmektedir. 8.

satırda, tespit edilen oyun karesinin içerisindeki bilgi piece değişkenine aktarılmaktadır. Dokuzuncu satırda ise bu bilgi daha önce belirlenen PceChar dizisi içerisinde hangi karaktere denk geliyorsa o karakter önüne ve arkasına boşluk karakteri eklenerek line değişkenine eklenmektedir. 10. satırda, içerideki for döngüsünden çıkılmakta ve 11. satırda line değişkeni konsola yazdırılmaktadır. Bu işlem tüm satırlar için dıştaki for döngüsü aracılığıyla tekrarlanmaktadır. 15 ve 17. satırlar arasında yazdırılan oyun tahtasının sütun etiketleri yazdırılmaktadır. 19. satırda sırasıyla beyaz tarafa ve siyah tarafa ait geçerken alma kareleriyle ilgili bilgiler yazdırılmaktadır. Şayet herhangi bir geçerken alma karesi mevcut değilse 99 değeri yazdırılmaktadır. 21 ve 25. satırlar arasında rok bilgisi yazdırılmaktadır. 21 ve 25. satırlar arasında kullanılan if yapılarında 0'dan farklı her değer dönüşü true, 0 ise false değerini temsil etmektedir. 26. satırda ise pozisyon anahtarı bilgisi yazdırılmaktadır. Okunması daha kolay olduğu için bu bilgi 16'lık sayı tabanında yazdırılmaktadır.

3.11 Oyun Taşı Listesinin Konsola Alfa-Nümerik Olarak Yazdırılması

Hata ayıklama (debug) işlemlerinde kullanılmak üzere oyun taşlarının bulunduğu oyun karesi a1, a2 gibi alfa-nümerik olarak yazdırılmaktadır. Nümerik olarak verilen bir oyun karesi aşağıdaki **PrSq** fonksiyonu kullanılarak alfa-nümerik hale getirilmektedir. Fonksiyona parametre olarak verilen oyun karesinin sütun numarası FilesBrd[sq] dizisi ile elde edilmekte ve sonucu FileChar dizisine indeks olarak verilmektedir. FileChar dizisinde o indekse karşılık gelen karakter elde edilmektedir. Benzer işlem satır karakterini elde etmek için de kullanılmaktadır. Elde edilen bu iki karakter yan yana getirilerek fonksiyon dönüşü sağlanmaktadır.

```
function PrSq(sq) {
    return (FileChar[FilesBrd[sq]] + RankChar[RanksBrd[sq]]);
}
```

Oyun taşlarının bulunduğu karenin alfa-nümerik değerleri aşağıdaki fonksiyon kullanılarak konsol ekranına yazdırılmaktadır;

```
1  function PrintPieceLists() {
2      var piece, pceNum;
3      for(piece = PIECES.wP; piece <= PIECES.bK; ++piece) {
```

```
4         for(pceNum = 0; pceNum < GameBoard.pceNum[piece]; ++pceNum) {
5             console.log('Piece ' + PceChar[piece] + ' on ' +
              PrSq( GameBoard.pList[PCEINDEX(piece,pceNum)] ));
6         }
7     }
8 }
```

Üçüncü satırda tüm oyun taşlarını gezecek şekilde bir for döngüsü başlatılmaktadır. Dördüncü satırda ise ilk for döngüsünde belirlenen tipteki bütün taşları gezecek şekilde ikinci bir for döngüsü başlatılmaktadır. Beşinci satırda ise ilgili oyun taşının hangi oyun karesinde olduğu konsol ekranına yazdırılmaktadır.

3.12 Oyun Karesinin Saldırı Altında Olup Olmadığının Kontrolü

Belirli bir oyun pozisyonunda yapılabilecek hamlelerin listesini çıkaracak olan fonksiyonu kodlamadan önce bu fonksiyona yardımcı olacak olan bir oyun karesinin tehdit/saldırı altında olup olmadığı kontrolünü yapan bir fonksiyonun hazırlanması gerekmektedir. Bu fonksiyon sayesinde şahın bir kareye hareket edip edemeyeceği bilgisi elde edilebilmektedir.

Bu fonksiyon içerisinde kullanılan bazı diziler aşağıdaki şekilde defs.js dosyası içerisinde tanımlanmaktadır;

```
var KnDir = [ -8, -19,   -21, -12, 8, 19, 21, 12 ];
var RkDir = [ -1, -10,   1, 10 ];
var BiDir = [ -9, -11, 11, 9 ];
var KiDir = [ -1, -10,    1, 10, -9, -11, 11, 9 ];
```

Yukarıda verilen dizilerde bir oyun karesine hangi yönlerden saldırı olabileceği bilgisi yer almaktadır. KnDir dizisinde bir atın hangi yönlerden bir kareye saldırabileceği verilmektedir. Örneğin Şekil 3-3'deki 56. karede beyaz piyon olduğunu düşünelim. Bu durumda karşı tarafın yani, siyah tarafın atının saldırısı altında olup olmadığının kontrolü için 56. karenin etrafında mavi ile belirtilmiş olan karelerin kontrol edilmesi gerekmektedir. Bu da KnDir dizisi kullanılarak sağlanmaktadır. Örneğin; 56 sayısını dizinin ilk elemanı ile topladığımızda; 56 + (-8) = 48 sonucunu verecektir. 48 de mavi ile işaretlenmiş karelerden birisidir. Dizinin diğer elemanları da diğer mavi karelerin tespitinde kullanılmaktadır. Yukarıda tanımlanan diğer diziler de aynı şekilde kullanılmaktadır. RkDir dizisi kale ile yapılacak saldırılarda saldırının geleceği

yönlerin tespitinde, BiDir dizisi fil tarafından yapılacak saldırının yönlerinin tespitinde, Kidir ise şah ile yapılacak saldırının yönlerinin tespitinde kullanılmaktadır. Vezirin yapacağı saldırıların tespitinde ise fil ve kalenin birleşimi kullanılmaktadır.

0	1	2	3	4	5	6	7	8	9	
10	11	12	13	14	15	16	17	18	19	
20	wN	22	23	24	25	26	27	28	29	A1 == 21
30	31	32	33	34	35	36	37	38	39	
40	41	42	43	44	45	46	47	48	49	
50	51	52	53	54	55	56	57	58	59	
60	61	62	63	64	65	66	67	68	69	
70	71	72	73	74	75	76	77	78	79	
80	81	82	83	84	85	86	87	88	89	
90	91	92	93	94	95	96	97	98	99	H8 == 98
100	101	102	103	104	105	106	107	108	109	
110	111	112	113	114	115	116	117	118	119	

Şekil 3-3 Bir oyun karesine saldırı

Bir oyun karesinin saldırı altında olup olmadığı aşağıdaki **SqAttacked** fonksiyonu ile tespit edilmektedir;

```
1  function SqAttacked(sq, side) {
2      var pce;
3      var t_sq;
4      var index;
5
6      if(side == COLOURS.WHITE) {
7          if(GameBoard.pieces[sq - 11] == PIECES.wP ||
           GameBoard.pieces[sq - 9] == PIECES.wP) {
8              return BOOL.TRUE;
9          }
10     } else {
11         if(GameBoard.pieces[sq + 11] == PIECES.bP ||
           GameBoard.pieces[sq + 9] == PIECES.bP) {
12             return BOOL.TRUE;
13         }
14     }
15
16     for(index = 0; index < 8; index++) {
17         pce = GameBoard.pieces[sq + KnDir[index]];
18         if(pce != SQUARES.OFFBOARD && PieceCol[pce] ==
           side && PieceKnight[pce] == BOOL.TRUE) {
```

```
19              return BOOL.TRUE;
20          }
21      }
22
23      for(index = 0; index < 4; ++index) {
24          dir = RkDir[index];
25          t_sq = sq + dir;
26          pce = GameBoard.pieces[t_sq];
27          while(pce != SQUARES.OFFBOARD) {
28              if(pce != PIECES.EMPTY) {
29                  if(PieceRookQueen[pce] == BOOL.TRUE &&
                    PieceCol[pce] == side) {
30                      return
                        BOOL.TRUE;
31                  }
32                  break;
33              }
34              t_sq += dir;
35              pce = GameBoard.pieces[t_sq];
36          }
37      }
38
39      for(index = 0; index < 4; ++index) {
40          dir = BiDir[index];
41          t_sq = sq + dir;
42          pce = GameBoard.pieces[t_sq];
43          while(pce != SQUARES.OFFBOARD) {
44              if(pce != PIECES.EMPTY) {
45                  if(PieceBishopQueen[pce] == BOOL.TRUE &&
                    PieceCol[pce] == side) {
46                      return
                        BOOL.TRUE;
47                  }
48                  break;
49              }
50              t_sq += dir;
51              pce = GameBoard.pieces[t_sq];
52          }
53      }
54
55      for(index = 0; index < 8; index++) {
56          pce = GameBoard.pieces[sq + KiDir[index]];
57          if(pce != SQUARES.OFFBOARD && PieceCol[pce] == side &&
            PieceKing[pce] == BOOL.TRUE) {
58              return BOOL.TRUE;
59          }
```

```
60      }
61
62      return BOOL.FALSE;
63
64  }
```

SqAttacked fonksiyonunun 1. satırında fonksiyon sq ve side parametreleri verilerek başlatılmaktadır. sq parametresi ile saldırı olup olmadığını öğrenmek istediğimiz oyun karesi bilgisi girilmekte ve side ile de saldırı yapan tarafın rengi belirlenmektedir. 6. ve 9. kareler arasında saldırının beyaz piyon (lar) tarafından yapıldığı değerlendirilmektedir. Saldırı var mı yok mu diye sorgulanan karenin -9 ve -11. karelerine denk gelen karelerde yani, sağ çapraz ve sol çapraz önündeki karelerde beyaz piyon varsa BOOL.TRUE değeri döndürülmektedir. Yani, o karenin saldırı altında olduğu belirtilmektedir. Şayet saldıran taraf siyahsa 10. ve 14. satırlar arasındaki bölüm çalışmaktadır. Bu durumda saldıran taraf siyah taraf olduğu için bulunulan karenin +9 ve +11 karelerine denk gelen karelerde siyah piyon olup olmadığı kontrol edilmektedir. Şekil 3-4'de bu durum gösterilmektedir. 35. karede bir beyaz oyun taşı bulunduğunu düşündüğümüzde ve bu kareye siyah tarafın piyonları tarafından bir saldırı olup olmadığını görmek için sq + 9 = 35 + 9 = 44 ve sq + 11 = 35 + 11 = 46. kareleri kontrol etmek gerekmektedir.

0	1	2	3	4	5	6	7	8	9	
10	11	12	13	14	15	16	17	18	19	
20	21	22	23	24	25	26	27	28	29	A1 == 21
30	31	32	33	34	35	36	37	38	39	
40	41	42	43	44	45	46	47	48	49	
50	51	52	53	54	55	56	57	58	59	
60	61	62	63	64	65	66	67	68	69	
70	71	72	73	74	75	76	77	78	79	
80	81	82	83	84	85	86	87	88	89	
90	91	92	93	94	95	96	97	98	99	H8 == 98
100	101	102	103	104	105	106	107	108	109	
110	111	112	113	114	115	116	117	118	119	

Şekil 3-4 Siyah piyon saldırısı

16 ve 21. satırlar arasında bir at tarafından saldırıya uğranıp uğranmadığı tespit edilmektedir. Bir kareye 8 farklı yönden at saldırısı ihtimali olduğundan dolayı 16. satırdaki for döngüsünün tekrar değeri 8 olarak belirlenmektedir. 17. satırda saldırı olup olmadığı tespit edilmeye çalışılan karenin etrafında at ile saldırı yapabilecek olan karelerdeki taşlar belirlenmektedir. Şayet bu taşlar saldırı

yapan tarafa ait at taşları ise BOOL.TRUE değeri döndürülmektedir (18 - 20. satırlar). 23 ve 37. satırlar arasında bir karenin kale veya vezirin (yatay ve dikey hamleler için) saldırısı altında olup olmadığı tespit edilmektedir. Yatay ve dikey hamlelerle saldırı dört farklı noktadan gelebileceği için 23. satırda for döngüsünün tekrar miktarı dört olarak belirlenmektedir. 24. satırda saldırının hangi yönden geldiği dir değişkenine aktarılmakta ve 25. satırda bu değer saldırı olup olmadığı belirlenmek istenilen karenin değerine eklenmektedir. Böylelikle saldırının gelebileceği yönlere doğru ilk kare belirlenmiş olmaktadır (t_sq). Bu da o karenin sağ, sol, üst veya altında bulunan karelerden biri olmaktadır. 26. satırda ise t_sq karesindeki oyun taşı değeri pce değişkenine atanmaktadır. 27. satırda bir while döngüsü başlatılmakta ve pce değeri SQUARES.OFFBOARD'den faklı olduğu sürece çalışmaktadır. Yani, oyun tahtasının dışına çıkılmadığı sürece bu döngü çalışmaktadır. 28. satırda pce değeri boş bir değer değilse bir if yapısının içerisine girilmektedir. 29. satırda bu pce değerinin kale veya vezir olup olmadığı ve aynı zamanda pce'in renginin verilen side değeriyle aynı olup olmadığı kontrol edilmektedir. Şayet aynı değerde ise BOOL.TRUE dönüşü yapılmaktadır (30. satır). 29. satırdaki if sorgusunun içerisine girme koşulu oluşmamışsa karşılaşılan taş saldırı olup olmadığı sorgulanan taşla aynı renkte demektir. Dolayısıyla while döngüsünden çıkılmaktadır (32. satır). Kontrol edilen karenin boş bir kare olması durumunda 34. satıra geçilmekte ve t_sq dir kadar arttırılarak o yöndeki bir sonraki kareye geçilmektedir. 35. satırda bu karedeki oyun taşının değeri pce değişkenine aktarılmakta ve while döngüsünün başına dönülmektedir. Böylelikle o yöndeki saldırı gelebilecek olan tüm kareler kontrol edilmektedir. 36. satırda while döngüsünden çıkma şartları sağlandığında while döngüsünden çıkılmakta ve 37. satırda for döngüsünün başına dönülmektedir. Diğer üç yön için de aynı şekilde işlemler yapılarak for döngüsü tamamlanmaktadır. 39. satırda fil ve vezir (sadece çapraz yönde hamleler) saldırılarının tespiti için dört tekrarlı bir for döngüsü başlatılmakta ve 53. satırda sonlandırılmaktadır. Bu döngü de bir önceki döngü ile benzer şekilde çalışmaktadır. 55. satırda şah tarafından yapılan bir saldırı olup olmadığının tespiti için 8 tekrarlı bir for döngüsü başlatılmaktadır. Bu döngüde verilen sq değerinin etrafındaki tüm kareler kontrol edilmektedir. Etraftaki bu karelerin içerisinde değer 56. satırda pce değişkenine aktarılmaktadır. 57. satırda etraftaki bu karelerin boş olmaması, side parametresi ile verilen tarafla aynı tarafta olması ve taşın değerinin şah olması durumunda if yapısının içerisine girilmektedir. Şartlar sağlandığı için de 58. satırda BOOL.TRUE değeri dönülmektedir. Bu işlem sq karesinin etrafındaki tüm yönlerdeki kareler için tekrarlanmaktadır. Şayet buraya kadar hiç bir dönüş olmazsa BOOL.FALSE değeri 62. satırda döndürülmektedir ve 64. satırda fonksiyon sonlandırılmaktadır.

3.13 Hamle Değişkeninin Yapısı

Bir hamle yapılırken bir oyun taşının nereden nereye hareket ettiği ile beraber bazı ekstra bilgilerin de kaydının tutulması gerekmektedir. Bu bilgiler aşağıdaki gibidir;

Nereden hareket ediliyor? (FromSq)

Nereye hareket ediliyor? (ToSq)

Yenilen taş var mı? (Captured Piece)

Geçerken alma karesine hareketle yenilen bir piyon var mı? (EnPass Capture)

Piyon başlangıç pozisyonunda iken iki kare ileri mi hareket etti? (Pawn Start)

Bir piyon terfi edeceği kareye ulaştı mı? (Promoted Piece)

Rok hamlesi yapıldı mı? (Castling Move)

Bu bilgilerinin tamamının ayrı ayrı değişkenlerde tutulmasının kodlamayı zorlaştıracağı düşünüldüğünden dolayı hamle bilgilerinin hepsi bir integer değişken içerisinde toplanmaktadır. Bu işlem için de bitsel işlemler kullanılmaktadır. Bu bitsel işlemleri daha iyi anlaşılması için her bir bit'in bulunduğu konumda taşıdığı ondalık değer Tablo 3-1'de verilmektedir.

Tablo 3-1 İkili tabanda bitlerin taşıdığı onluk değerler

Bitler	8. bit	7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit
Onluk Değer	128	64	32	16	8	4	2	1

Bir hamle bilgisi oluşturmak için bir integer'ın ilk 28 biti kullanılmaktadır. Oyun karesi bilgisi 21 - 98 sayıları aralığında değişmekte ve 7 bitle temsil edilebilmektedir. 28 bitin en düşük değere sahip (en sağdaki bitler) bitleri FromSq bilgisini tutmak için kullanılmaktadır. Örneğin elimizde bir hamlenin tutulduğu Move değişkeni olsun ve değeri de 0 olsun. FromSq bilgisi de 98 olsun. Bu bilgiyi Move değişkenine aktarmak için bitsel VEYA (bitwise OR) işlemi yapılmaktadır. Bu örneğin bitsel gösterimi aşağıdaki şekildedir;

Move = 0000 0000 0000 0000 0000 0000 0000
OR

FromSq = 0000 0000 0000 0000 0000 0110 0010
Move = 0000 0000 0000 0000 0000 0110 0010

Move değişkenine ToSq bilgisini eklemek için ise ToSq bilgisi 7 bit sola kaydırıldıktan sonra Move değişkeniyle OR işlemi yapılmaktadır. ToSq'in 88 olduğunu varsayalım.

ToSq = 0000 0000 0000 0000 0000 0101 1000 << 7
ToSq = 0000 0000 0000 0010 1100 0000 0000
OR
Move = 0000 0000 0000 0000 0000 0110 0010
Move = 0000 0000 0000 0010 1100 0110 0010

Son durumda Move değişkeni hem FromSq hem de ToSq bilgilerini barındırmaktadır. Buna ilave olarak yenilen taş olup olmadığı bilgisinin de eklenmesi benzer şekilde yapılmaktadır. Taşlar 1 - 12 arasında numaralandırıldığı için 4 bitlik bir alan bunun için yeterli olmaktadır. Hamle sırasında yenilen bir taş varsa bunun bilgisi 14 bit sola kaydırılmakta ve Move değişkeni ile OR işlemine tabii tutulmaktadır. Şayet yapılan hamle bir geçerken alma hamlesiyse (Enpassant Capture) bu durum bir bitle belirtilmektedir. Bunun için de Move değişkenini 0x40000 (Hex) sayısıyla OR işlemine tabii tutulmaktadır. Benzer şekilde hamle bir piyonun başlangıçta iki kare ileri gitme hamlesiyse (Pawn start) bunu belirtmek için de Move değişkeni 0x80000 (Hex) değeriyle OR işlemine tabii tutulmaktadır. Bir piyonun terfi sonucu hangi taşa terfi ettiği ise terfi edilen taş değeri 20 bit sola kaydırılarak Move değişkenine OR işlemi ile eklenmektedir. Yapılan hamle bir rok hamlesi ise bu bilgi bir bitle temsil edilmektedir. 0x1000000 değeri Move değişkeni ile OR işlemine tabii tutularak bu bilgi Move değişkenine eklenmektedir.

Yukarıda onaltılık sayı tabanında verilen sayısal değerler flag (bayrak) değerleri olarak defs.js içerisinde aşağıdaki şekilde tanımlanmaktadır;

```
var MFLAGEP = 0x40000;
var MFLAGPS = 0x80000;
var MFLAGCA = 0x1000000;
```

Ayrıca bir taş yeme işlemi olup olmadığını belirlemek için aşağıdaki tanımlama defs.js dosyası içerisinde yer almaktadır;

var MFLAGCAP = 0x7C000;

Bu değer Move değişkeni ile AND işlemine tabii tutulur ve sıfırdan farklı bir değer elde edildiğinde yapılan hamle ile bir taş yeme işlemi gerçekleştirildiği görülmektedir. Bu değerin ikili sayı tabanındaki karşılığı ile karşılaştırması aşağıdaki Tablo 3-2'deki gibidir;

Tablo 3-2 Taş yeme işlemi bayrağının (MFLAGCAP) on altılı ve ikili sayı tabanlarında gösterimi

On altılı Taban	0	0	7	C	0	0	0
İkili Taban	0000	0000	0111	1100	0000	0000	0000

Tablo 3-2'de ikili taban gösteriminde sağdan 15-18. bitler Move değişkeninde yenilen taşın bilgilerinin tutulduğu alandaki veri için bayrak görevi görmektedir. Sağdan 19. bit ise geçerken alma işlemi için bayrak görevi görmektedir. Sonuçta her iki tür bilgi de bir yeme işlemini gösterdiği için her iki alan da bu bayrak içerisine dahil edilmektedir.

Bir terfi olup olmadığını tespit etmek için de aşağıdaki değişken defs.js içerisinde yer almaktadır;

var MFLAGPROM = 0xF00000;

Herhangi bir hamle yapılmadığını gösteren sabit olan "var NOMOVE = 0;" tanımı defs.js içerisinde yer almaktadır.

Bu bölümde şu ana kadar bahsedilen bir hamle ile ilgili bilgilerin bir integer'da birleştirme işlemi movegen.js dosyası içerisinde aşağıdaki şekilde kodlanmaktadır;

```
function MOVE(from, to, captured, promoted, flag) {
  return (from | (to << 7) | (captured << 14) | (promoted << 20) | flag);
}
```

Yukarıdaki fonksiyonda from ve flag parametreleri için herhangi bir bitsel kaydırma işlemi yapılmamaktadır. to parametresi 7 bit, captured parametresi 14 bit ve promoted parametresi 20 bit sola kaydırılmaktadır. Ardından da hepsi OR

işlemine tabi tutulmakta ve elde edilen değer fonksiyon çıktısı olarak döndürülmektedir.

Hamle değişkeninin içerisindeki bilgiler birbirinden ayrılıp kullanılmak istendiğinde ise tam tersi işlem yapmak gerekmektedir. Yani, sola kaydırma işlemi kadar sağa kaydırma işlemi yapılmalı ve ardından elde edilmek istenilen bilginin kapladığı bit alanı kadar "1" değeri içeren bir sayı ile AND işlemine tabii tutulması gerekmektedir. ToSq değeri daha önce oluşturulan Move değişkeninden aşağıdaki şekilde çıkarılmaktadır;

Move = 0000 0000 0001 10**10 1100 0**110 0010 >> 7
Move = 0000 0000 0000 0000 0110 0**101 1000**
AND
0000 0000 0000 0000 0000 **0111 1111** (0x7F (Hex))
Sonuç (ToSq) = 0000 0000 0000 0000 0000 0**101 1000**

Hamle değişkeni içerisindeki değerlerin elde edilmesi defs.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
function FROMSQ(m) { return (m & 0x7F); }
function TOSQ(m) { return ( (m >> 7) & 0x7F); }
function CAPTURED(m) { return ( (m >> 14) & 0xF); }
function PROMOTED(m) { return ( (m >> 20) & 0xF); }
```

FROMSQ fonksiyonunda m hamle parametresi 0x7F (ikili tabanda : 1111111) sayısıyla AND (&) işlemine tabii tutulmaktadır ve FromSq en sağdaki 7 bit olduğundan dolayı sonuç olarak FromSq elde edilmektedir. **TOSQ** fonksiyonunda m hamle parametresi 7 bit sağa kaydırıldıktan sonra 0x7F sayısıyla AND (&) işlemine tabii tutularak sonuca ulaşılmaktadır. **CAPTURED** fonksiyonunda m hamle parametresi 14 bit sağa kaydırıldıktan sonra 0xF (ikili tabanda : 1111) sayısıyla AND işlemine tabii tutularak hangi taşın yendiği bilgisi elde edilmektedir. **PROMOTED** fonksiyonunda m hamle parametresi 20 bit sağa kaydırıldıktan sonra 0xF sayısıyla AND işlemine tabii tutularak hangi taşa terfi edildiği bilgisi elde edilmektedir.

3.14 Hamle Listesinin Oluşturulması

Mevcut pozisyonuna ait yapılabilecek tüm hamlelerin listesi **GenerateMoves** fonksiyonu kullanılarak oluşturulmaktadır. **GenerateMoves** fonksiyonu aşağıdaki şekilde movegen.js dosyası içerisinde yer almaktadır;

```
1   function GenerateMoves() {
2     GameBoard.moveListStart[GameBoard.ply+1] =
      GameBoard.moveListStart[GameBoard.ply];
3
4     var pceType;
5     var pceNum;
6     var sq;
7     var pceIndex;
8     var pce;
9     var t_sq;
10    var dir;
11
12    if(GameBoard.side == COLOURS.WHITE) {
13      pceType = PIECES.wP;
14
15      for(pceNum = 0;pceNum < GameBoard.pceNum[pceType];++pceNum){
16        sq = GameBoard.pList[PCEINDEX(pceType, pceNum)];
17
18        if( GameBoard.pieces[sq + 10] == PIECES.EMPTY ) {
19          AddWhitePawnQuietMove(sq, sq+10);
20          if( (RanksBrd[sq] == RANKS.RANK_2) &&
            (GameBoard.pieces[sq + 20] == PIECES.EMPTY) ) {
21              AddQuietMove( MOVE(sq, sq + 20, PIECES.EMPTY,
                PIECES.EMPTY, MFLAGPS ));
22          }
23        }
24
25        if(SQOFFBOARD(sq + 9) == BOOL.FALSE &&
           PieceCol[GameBoard.pieces[sq+9]] == COLOURS.BLACK &&
          GameBoard.pieces[sq+9] != PIECES.bK) {
26          AddWhitePawnCaptureMove(sq, sq + 9, GameBoard.pieces[sq+9]);
27        }
28
29        if(SQOFFBOARD(sq + 11) == BOOL.FALSE &&
          PieceCol[GameBoard.pieces[sq+11]] == COLOURS.BLACK &&
          GameBoard.pieces[sq+11] != PIECES.bK) {
30          AddWhitePawnCaptureMove(sq, sq+11, GameBoard.pieces[sq+11]);
31        }
32
```

```
33          if(GameBoard.enPas[COLOURS.BLACK] != SQUARES.NO_SQ){
34              if(sq + 9 == GameBoard.enPas[1]) {
35                  AddEnPassantMove( MOVE(sq, sq+9, PIECES.EMPTY,
                    PIECES.EMPTY,
                    MFLAGEP ) );
36              }
37
38              if(sq + 11 == GameBoard.enPas[1]) {
39                  AddEnPassantMove( MOVE(sq, sq+11, PIECES.EMPTY,
                    PIECES.EMPTY,
                     MFLAGEP ) );
40              }
41          }
42      }
43
44      if(GameBoard.castlePerm & CASTLEBIT.WKCA) {
45          if(GameBoard.pieces[SQUARES.F1] == PIECES.EMPTY &&
            GameBoard.pieces[SQUARES.G1] == PIECES.EMPTY) {
46              if(SqAttacked(SQUARES.F1, COLOURS.BLACK) ==
                BOOL.FALSE && SqAttacked(SQUARES.E1, COLOURS.BLACK)
                == BOOL.FALSE) {
47                  AddQuietMove( MOVE(SQUARES.E1, SQUARES.G1,
                    PIECES.EMPTY,
                    PIECES.EMPTY, MFLAGCA ));
48              }
49          }
50      }
51
52      if(GameBoard.castlePerm & CASTLEBIT.WQCA) {
53          if(GameBoard.pieces[SQUARES.D1] == PIECES.EMPTY &&
            GameBoard.pieces[SQUARES.C1] == PIECES.EMPTY &&
            GameBoard.pieces[SQUARES.B1] == PIECES.EMPTY) {
54              if(SqAttacked(SQUARES.D1, COLOURS.BLACK) ==
                BOOL.FALSE && SqAttacked(SQUARES.E1, COLOURS.BLACK)
                == BOOL.FALSE) {
55                  AddQuietMove( MOVE(SQUARES.E1, SQUARES.C1,
                    PIECES.EMPTY, PIECES.EMPTY, MFLAGCA ));
56              }
57          }
58      }
59
60  } else {
61      pceType = PIECES.bP;
62
63      for(pceNum = 0; pceNum < GameBoard.pceNum[pceType]; ++pceNum){
64          sq = GameBoard.pList[PCEINDEX(pceType, pceNum)];
65
```

```
66          if( GameBoard.pieces[sq - 10] == PIECES.EMPTY ) {
67              AddBlackPawnQuietMove(sq, sq-10);
68              if( RanksBrd[sq] == RANKS.RANK_7 &&
                GameBoard.pieces[sq - 20] == PIECES.EMPTY ){
69                  AddQuietMove( MOVE(sq, sq - 20, PIECES.EMPTY,
                    PIECES.EMPTY, MFLAGPS ));
70              }
71          }
72
73          if(SQOFFBOARD(sq - 9) == BOOL.FALSE &&
            PieceCol[GameBoard.pieces[sq-9]] == COLOURS.WHITE &&
            GameBoard.pieces[sq-9] != PIECES.wK) {
74               AddBlackPawnCaptureMove(sq, sq - 9, GameBoard.pieces[sq-9]);
75          }
76
77          if(SQOFFBOARD(sq - 11) == BOOL.FALSE &&
            PieceCol[GameBoard.pieces[sq-11]] == COLOURS.WHITE &&
            GameBoard.pieces[sq-11] != PIECES.wK) {
78              AddBlackPawnCaptureMove(sq, sq - 11, GameBoard.pieces[sq-11]);
79          }
80
81          if(GameBoard.enPas[COLOURS.WHITE] != SQUARES.NO_SQ) {
82              if(sq - 9 == GameBoard.enPas[0]) {
83                  AddEnPassantMove( MOVE(sq, sq-9, PIECES.EMPTY,
                    PIECES.EMPTY, MFLAGEP ) );
84              }
85
86              if(sq - 11 == GameBoard.enPas[0]) {
87                  AddEnPassantMove( MOVE(sq, sq-11, PIECES.EMPTY,
                    PIECES.EMPTY, MFLAGEP ) );
88              }
89          }
90      }
91
92      if(GameBoard.castlePerm & CASTLEBIT.BKCA) {
93          if(GameBoard.pieces[SQUARES.F8] == PIECES.EMPTY &&
            GameBoard.pieces[SQUARES.G8] == PIECES.EMPTY) {
94              if(SqAttacked(SQUARES.F8, COLOURS.WHITE) ==
                BOOL.FALSE && SqAttacked(SQUARES.E8, COLOURS.WHITE)
                == BOOL.FALSE) {
95                  AddQuietMove( MOVE(SQUARES.E8, SQUARES.G8,
                    PIECES.EMPTY, PIECES.EMPTY, MFLAGCA ));
96              }
97          }
98      }
99
100     if(GameBoard.castlePerm & CASTLEBIT.BQCA) {
```

```
101         if(GameBoard.pieces[SQUARES.D8] == PIECES.EMPTY &&
            GameBoard.pieces[SQUARES.C8] == PIECES.EMPTY &&
            GameBoard.pieces[SQUARES.B8] == PIECES.EMPTY) {
102           if(SqAttacked(SQUARES.D8, COLOURS.WHITE) ==
              BOOL.FALSE && SqAttacked(SQUARES.E8, COLOURS.WHITE)
              == BOOL.FALSE) {
103             AddQuietMove( MOVE(SQUARES.E8, SQUARES.C8,
                PIECES.EMPTY, PIECES.EMPTY, MFLAGCA ));
104           }
105         }
106       }
107     }
108
109     pceIndex = LoopNonSlideIndex[GameBoard.side];
110     pce = LoopNonSlidePce[pceIndex++];
111
112     while (pce != 0) {
113       for(pceNum = 0; pceNum < GameBoard.pceNum[pce]; ++pceNum) {
114         sq = GameBoard.pList[PCEINDEX(pce, pceNum)];
115
116         for(index = 0; index < DirNum[pce]; index++) {
117           dir = PceDir[pce][index];
118           t_sq = sq + dir;
119
120           if(SQOFFBOARD(t_sq) == BOOL.TRUE) {
121               continue;
122           }
123
124           if(GameBoard.pieces[t_sq] == Kings[PieceCol[pce]^1]) continue;
125           var IsAttacked = SqAttacked(t_sq, GameBoard.side^1);
126
127           if(  pce == Kings[PieceCol[pce]] && IsAttacked == BOOL.TRUE ){
128             continue;
129           }
130
131           if(GameBoard.pieces[t_sq] != PIECES.EMPTY ) {
132             if(PieceCol[GameBoard.pieces[t_sq]] != GameBoard.side) {
133               AddCaptureMove( MOVE(sq, t_sq,
                  GameBoard.pieces[t_sq], PIECES.EMPTY, 0 ));
134             }
135           }else {
136             AddQuietMove( MOVE(sq, t_sq, PIECES.EMPTY,
                PIECES.EMPTY, 0 ));
137           }
138         } //end of for
139       }
```

```
140         pce = LoopNonSlidePce[pceIndex++];
141     }
142
143     pceIndex = LoopSlideIndex[GameBoard.side];
144     pce = LoopSlidePce[pceIndex++];
145
146     while(pce != 0) {
147         for(pceNum = 0; pceNum < GameBoard.pceNum[pce]; ++pceNum) {
148           sq = GameBoard.pList[PCEINDEX(pce, pceNum)];
149
150           for(index = 0; index < DirNum[pce]; index++) {
151               dir = PceDir[pce][index];
152               t_sq = sq + dir;
153
154               while(SQOFFBOARD(t_sq) == BOOL.FALSE){
155                   if(GameBoard.pieces[t_sq] == Kings[PieceCol[pce]^1]) break;
156                   if(GameBoard.pieces[t_sq] != PIECES.EMPTY ) {
157                     if(PieceCol[GameBoard.pieces[t_sq]] != GameBoard.side) {
158                       AddCaptureMove( MOVE(sq, t_sq,
                          GameBoard.pieces[t_sq],PIECES.EMPTY, 0 ));
159                       }
160                       break;
161                   }
162                   AddQuietMove( MOVE(sq, t_sq, PIECES.EMPTY,
                      PIECES.EMPTY, 0 ));
163                   t_sq += dir;
164               }
165           }
166       }
167       pce = LoopSlidePce[pceIndex++];
168     }
169 }
```

GenerateMoves fonksiyonunun ikinci satırında GameBoard.moveListStart dizisinin bir önceki indeksindeki değer bir sonraki indeksine aktarılmaktadır. Bu dizi hamlelerin kaydedildiği GameBoard.moveList dizisinin indekslemesini yapmaktadır. GameBoard.moveListStart dizisi, hangi GameBoard.ply değerinde (arama ağacı derinliğinde) nereden başlanacağı bilgisini tutmaktadır. Aşağıdaki şekilde çalışmaktadır;

GameBoard.ply = 0; ve GameBoard.moveListStart[GameBoard.ply]=0; değerleriyle ilkleme yapılmakta ve fonksiyon içerisinde aşağıdaki atama yapılmaktadır;

GameBoard.moveListStart[GameBoard.ply+1] =
GameBoard.moveListStart[GameBoard.ply];

Bu atamada indeks değerlerinin yerine konmuş hali;

GameBoard.moveListStart[1] = GameBoard.moveListStart[0];

Yani, GameBoard.moveListStart[1] = 0; şeklindedir. GameBoard.moveListStart[GameBoard.ply+1]'deki değer de başka fonksiyonlar tarafından GameBoard.moveList dizisinin indeks değeri olarak aşağıdaki şekilde kullanılmaktadır;

GameBoard.moveList[GameBoard.moveListStart[GameBoard.ply+1]] = move;

Örneğimizde bunun anlamı; GameBoard.moveList[0] = move; olmaktadır. Yani GameBoard.moveList dizisinin ilk indeksine bir hamle kaydedilmektedir. Bu atamanın ardından GameBoard.moveListStart[GameBoard.ply+1]++; yapılmaktadır. Yani, GameBoard.moveListStart[1]=1; yapılmaktadır. 10 adet hamle kaydı yapıldığını varsaydığımızda son hamle GameBoard.moveList dizisinin 9. indeksine kaydedilmekte ve GameBoard.moveListStart[1]'in değeri de 10 yapılmaktadır. Böylelikle bir sonraki kaydın nereye yapılacağı bilinmektedir. Bir sonraki GameBoard.ply değerine geçtiğimizde yani, GameBoard.ply değerini 1 yaptığımızda **GenerateMoves** fonksiyonu tekrar çağrılmakta ve GameBoard.moveListStart[GameBoard.ply+1] = GameBoard.moveListStart[GameBoard.ply]; ataması tekrar yapılmaktadır. Bu durumda GameBoard.moveListStart[2] = 10 olmaktadır. GameBoard.moveListStart[2]'de GameBoard.moveList için indeks olarak kullanılacağından dolayı dizide kalınan yerden kayıt yapılmaya devam edilmektedir. 4. ve 10. satırlar arasında bu fonksiyon içerisinde kullanılacak olan değişken tanımlamaları yapılmaktadır. 12. satırda GameBoard.side'ın COLOURS.WHITE olup olmadığı yani oynayan tarafın beyaz olup olmadığı sorgulanmaktadır. Şayet beyaz ise if yapısının içerisine girilmektedir. Bu fonksiyon içerisinde GameBoard.side değiştirilerek her iki taraf için de yapılabilecek hamleler üretilmektedir. 13. satırda pceType değişkenine PIECES.wP (beyaz piyon) değeri atanmaktadır. Bu işlem, o taş tipine ait geçerli hamleleri bulup kaydetmek için yapılmaktadır. 15. satırda mevcut pcType çeşidinden (beyaz piyon) ne kadar taş varsa (pceNum) o uzunlukta bir for döngüsü başlatılmaktadır. 16. satırda beyaz piyonun bulunduğu oyun karesi sq

değişkenine atanmakta ve bu işlem tüm beyaz piyonlar için döngü süresince yapılmaktadır. 18. satırda sq değişken değerinin +10'uncu karesinin boş olup olmadığı kontrol edilerek o piyonun bulunduğu yerin önündeki karenin boş olup olmadığı belirlenmektedir. Boşsa bir kare ileri hareket etme hamlesi oluşturarak GameBoard.moveList dizisine kaydetmek için 19. satırdaki **AddWhitePawnQuietMove(sq, sq+10)**; fonksiyonu çağrılmaktadır. Bu fonksiyon movegen.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function AddWhitePawnQuietMove(from, to) {
2    if(RanksBrd[from]==RANKS.RANK_7) {
3      AddQuietMove(MOVE(from,to,PIECES.EMPTY,PIECES.wQ,0));
4      AddQuietMove(MOVE(from,to,PIECES.EMPTY,PIECES.wR,0));
5      AddQuietMove(MOVE(from,to,PIECES.EMPTY,PIECES.wB,0));
6      AddQuietMove(MOVE(from,to,PIECES.EMPTY,PIECES.wN,0));
7    } else {
8      AddQuietMove(MOVE(from,to,PIECES.EMPTY,PIECES.EMPTY,0));
9    }
10 }
```

İlk satırda **AddWhitePawnQuietMove** fonksiyonuna from ve to kareleri parametre olarak verilmektedir. İkinci satırda hamleye oyun tahtasının 7. satırından başlanıp başlanmadığı sorgulanmaktadır. Bu sorgu oyun tahtasının 7. satırından 8. satırına hamle yapan beyaz piyonun terfi ettirilmesi gerektiğinden dolayı yapılmaktadır. Şayet sorgu olumluysa 3. ve 6. satırlar arasındaki kodlar çalıştırılarak dört farklı hamle oluşturulmaktadır. Bunun sebebi ise terfi edilebilecek olan wQ, wR, wB ve wN gibi dört farklı oyun taşı olmasıdır. 3. ve 6. satırlar arasında **AddQuietMove** fonksiyonuna parametre olarak **MOVE** fonksiyonunun çıktısı verilmektedir. **MOVE** fonksiyonuna ise parametre olarak from(sq), to(sq+10), captured (PIECES.EMPTY), promoted (PIECES.wQ/wR/wB/wN), flag (0) parametre olarak verilmektedir. Şayet 2. satırdaki sorgu sonucu olumsuzsa 8. satırdaki kod çalıştırılmaktadır. Burada verilen parametrelerle bir beyaz piyonun oyun tahtasının 7. satırından 8. satırına olan hamleleri dışındaki bir kare ileri hamleleri oluşturulmaktadır. Fonksiyonun 3.,4.,5.,6. ve 8. satırlarda çalıştırılan **AddQuietMove** fonksiyonu movegen.js dosyasında aşağıdaki şekilde yer almaktadır;

```
1  function AddQuietMove(move) {
2    GameBoard.moveList[GameBoard.moveListStart[GameBoard.ply+1]]
     = move;
3    GameBoard.moveScores[GameBoard.moveListStart[GameBoard.ply+1]]
```

```
      =  0;
 4    if(move == GameBoard.searchKillers[GameBoard.ply]) {
 5      GameBoard.moveScores[GameBoard.moveListStart[GameBoard.ply+1]]
        = 900000;
 6    } else if( move ==
      GameBoard.searchKillers[GameBoard.ply + MAXDEPTH] ) {
 7      GameBoard.moveScores[GameBoard.moveListStart[GameBoard.ply+1]]
        = 800000;
 8    } else {
 9       GameBoard.moveScores[GameBoard.moveListStart[GameBoard.ply+1]]
        = GameBoard.searchHistory[GameBoard.pieces[FROMSQ(move)] *
        BRD_SQ_NUM + TOSQ(move)];
10    }
11    GameBoard.moveListStart[GameBoard.ply+1]++;
12  }
```

AddQuietMove fonksiyonunun ilk satırında ilgili hamlenin kaydedildiği move değişkeni parametre olarak alınarak fonksiyon başlatılmaktadır. 2. satırda move değişkeni GameBoard.moveList dizisinin ilgili indeks alanına kaydedilmektedir. 3. ve 10. satırlar arasında en iyi pozisyonu arama algoritması (alpha beta algoritması) tarafından kullanılan hamlenin değeri GameBoard.moveScores dizisine kaydedilmektedir. Bu uygulamamada kullanılan arama fonksiyonları klasik satranca göre çalıştırıldığı için bu çalışmanın kapsamı dışında yer almaktadır. Bu nedenle arama fonksiyonlarına değinilmemektedir. 11. satırda GameBoard.moveListStart dizisinin ilgili indeksinin değeri bir arttırılarak bir sonraki kayıt için GameBoard.moveList dizisinin yeni indeksi belirlenmektedir.

Yukarıda açıklaması yapılan fonksiyonların çağrımından sonra **GenerateMoves** fonksiyonu 20. satırdan devam etmektedir. 20. satırda beyaz piyon, oyun tahtasının 2. satırındaysa ve iki kare ilerisi de boşsa (bir önündeki karenin boş olduğu bir üsteki if ile belirlenmişti (18.satır)) if yapısının içerisine girilmekte ve 21. satırdaki **AddQuiteMove** fonksiyonu MFLAGPS (piyon iki ileri hamle bayrağı) eklenerek çağrılmaktadır. Beyaz piyonun bulunduğu karenin sol çaprazı tahta dışı değilse, siyah renkte bir taş varsa ve o siyah taş şah değilse (PIECES.bK) (25. satır) **AddWhitePawnCaptureMove** fonksiyonu çağrılmaktadır (26. satır). Klasik satrançta şah karşı tarafın piyonunun tehdidinde kalmışsa bir sonraki hamlesinde bu tehditten kurtulmak için hamle yapmaktadır. Ancak eş zamanlı satranç oyununda her iki taraf da aynı anda hamle yaptığı için bir sonraki hamlede şah tehditten kaçarken piyonun şahı yeme hamlesi yapmasına olanak sağlanmaktadır. Ama şahın tahtadan çıkmaması gerektiği için de bu if

sorgusunda karşı tarafın şahına doğru yeme hamlesi yapılması engellenmektedir. **AddWhitePawnCaptureMove** fonksiyonuna parametre olarak from (sq), to(sq+9) ve captured(GameBoard.pieces[sq+9]) parametreleri verilmektedir. **AddWhitePawnCaptureMove** fonksiyonu movegen.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function AddWhitePawnCaptureMove(from, to, cap) {
2    if(RanksBrd[from]==RANKS.RANK_7) {
3        AddCaptureMove(MOVE(from, to, cap, PIECES.wQ, 0));
4        AddCaptureMove(MOVE(from, to, cap, PIECES.wR, 0));
5        AddCaptureMove(MOVE(from, to, cap, PIECES.wB, 0));
6        AddCaptureMove(MOVE(from, to, cap, PIECES.wN, 0));
7    } else {
8        AddCaptureMove(MOVE(from, to, cap, PIECES.EMPTY, 0));
9    }
10 }
```

AddWhitePawnCaptureMove fonksiyonunda da **AddWhitePawnQuietMove** fonksiyonunda olduğu gibi oyun tahtasının 7. satırından 8. satırına hamle yapılıp yapılmadığı kontrolü yapılmaktadır. Farklı olarak **AddQuietMove** yerine **AddCaptureMove** fonksiyonu çağrılmaktadır. **AddCaptureMove** fonksiyonuna parametre olarak verilen **MOVE** fonksiyonuna, cap (capture) parametresi ilave edilmektedir. **AddCaptureMove** fonksiyonu movegen.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function AddCaptureMove(move) {
2      GameBoard.moveList[GameBoard.moveListStart[GameBoard.ply+1]]
       = move;
3      GameBoard.moveScores[GameBoard.moveListStart[GameBoard.ply+1]++]
       = MvvLvaScores[CAPTURED(move) * 14 +
       GameBoard.pieces[FROMSQ(move)]] + 1000000;
4  }
```

AddCaptureMove fonksiyonunun 2. satırında hamle bilgisini içeren move değişkeni GameBoard.moveList dizisinde uygun indeks alanına aktarılmaktadır. 3. satırda, bu hamleye ait skor da GameBoard.moveScores dizisine aktarılmaktadır. **AddCaptureMove** ve **AddQuiteMove** fonksiyonlarının farkı; yapılan hamle skorlarının farklı şekilde ele alınmasından kaynaklanmaktadır. Çağrılan fonksiyonlardan tekrar **GenerateMoves** fonksiyonuna dönerek 29. satırdan devam edilmektedir. **GenerateMoves** fonksiyonunun 29. ve 31. satırları arasında bu sefer beyaz piyonun sağ çaprazı, sol çaprazı gibi analiz edilmekte ve

uygun hamle oluşturulmaktadır. 33. satırda siyah tarafın oluşturduğu bir geçerken alma karesi olup olmadığı sorgulanmaktadır. Şayet varsa if yapısının içerisine girilmektedir. Burada 34. ve 38. satırlarda iki farklı if sorgusuyla karşılaşılmaktadır. Bu sorgularda sırasıyla beyaz piyonun sol çaprazında mı sağ çaprazında mı geçerken alma karesi olduğu sorgulanmaktadır. Hangi taraftaysa o if yapısının içerisine girilerek 35. veya 39. satırdaki **AddEnPassantMove** fonksiyonu çağrılmaktadır. Bu fonksiyonun parametresi olan **MOVE** fonksiyonunun parametreleri from(sq) to(sq+9 veya sq+11), capture(PIECES.EMPTY), promote (PIECES.EMPTY) ve MFLAGEP şeklinde verilmektedir. Bu hamle sonucu her ne kadar bir taş yeme işlemi gerçekleşse de capture parametresi olarak boş kare değeri verilmektedir. Buradaki rakip piyonunun tahtadan çıkarılma işlemi **MakeMove** ve/veya **MakeSimMove** fonksiyonları tarafından ele alınmaktadır. **AddEnPassantMove** fonksiyonu movegen.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1 function AddEnPassantMove(move) {
2     GameBoard.moveList[GameBoard.moveListStart[GameBoard.ply+1]] =
      move;
3     GameBoard.moveScores
      [GameBoard.moveListStart[GameBoard.ply + 1]++] = 105 + 1000000;
4 }
```

Bu fonksiyonda yine standart şekilde move değişkeni GameBoard.moveList dizisine aktarılmaktadır. GameBoard.moveScores dizisine de hamleye ait skor aktarılmaktadır. **GenerateMoves** fonksiyonuna dönüldüğünde 44. satırdan devam edilmektedir. Buraya kadar olan bölümde beyaz piyonla ilgili tüm hamleler çıkarılmakta ve kaydedilmektedir. 44. satırdaki if sorgusunda GameBoard.castlePerm & CASTLEBIT.WKCA işlemi yapılmakta ve 0'dan farklı bir sonuç alındığında bu sonuç true kabul edilmekte ve if yapısının içerisine girilmektedir. GameBoard.castlePerm mevcut pozisyondaki tüm rok izinlerini içermektedir. CASTLEBIT.WKCA ise beyaz renge ait şah tarafındaki roku (kısa rok) temsil etmektedir ve "1" değerine sahiptir. 45. satırda kısa rok yapılacak istikametteki karelerin boş olup olmadığının kontrolü yapılmaktadır. Şayet boşsa 46. satırda beyaz şahın üzerinden geçeceği F1 karesi ve üzerinde bulunduğu E1 karesinin tehdit altında olup olmadığı kontrol edilmektedir. Şayet tehdit yoksa 47. satırdaki **AddQuietMove** fonksiyonu MFFLAGCA bayrağı parametre olarak eklenerek çağrılmaktadır. MFFLAGCA bayrağı **MakeMove** ve **MakeSimMove** fonksiyonlarında rok yapılan yöndeki kalenin hareket ettirilmesi için kullanılmaktadır. 52. ve 57. satırlar arasında beyaz vezir tarafında rok (uzun rok)

hamlesi yapılıp yapılamayacağı sorgulanmakta ve şayet yapılabilecekse gerekli hamleler oluşturulmaktadır. Şu ana kadar beyaz piyonun tüm hamleleri ve beyaz tarafın rok hamleleri kaydedilmiş bulunmaktadır. 60 ve 107. satırlar arasında siyah piyonun hamleleri ve rok hamleleri belirlenerek kaydedilmektedir. Siyah tarafın kodlaması da beyaz tarafın kodlaması gibidir ancak oyun tahtasındaki yönler değiştiği için hamle yapılacak olan karelerin tespiti de değişmektedir. Örneğin; beyaz tarafta bir ileri piyon hamlesi yaparken sq+10 karesine ilerlenmekteyken siyah tarafta sq-10 karesine hamle yapılması gerekmektedir. Rok ile ilgili hamlelerde de ise oyun tahtasının 8. satırında olunduğu için ve şahın ve vezirin bulunduğu sütunlar değiştiği için kodlama da uygun şekilde değiştirilmektedir. 109. ve 141. satırlar arasında kayarak ilerlemeyen at ve şah gibi taşların hamleleri tespit edilip kaydedilmektedir. Hem kayarak ilerleyen hem de kayarak ilerlemeyen taşların hamlelerinin tespitinde kullanılacak olan bazı diziler aşağıdaki şekilde defs.js dosyası içerisinde tanımlanmaktadır;

```
var KnDir = [ -8, -19, -21, -12, 8, 19, 21, 12 ];
var RkDir = [ -1, -10, 1, 10 ];
var BiDir = [ -9, -11, 11, 9 ];
var KiDir = [ -1, -10, 1, 10, -9, -11, 11, 9 ];
var DirNum = [ 0, 0, 8, 4, 4, 8, 8, 0, 8, 4, 4, 8, 8 ];
var PceDir = [ 0, 0, KnDir, BiDir, RkDir, KiDir, KiDir, 0, KnDir, BiDir, RkDir,
KiDir, KiDir ];
var LoopNonSlidePce = [ PIECES.wN, PIECES.wK, 0, PIECES.bN, PIECES.bK,
0 ];
var LoopNonSlideIndex = [ 0, 3 ];
var LoopSlidePce = [ PIECES.wB, PIECES.wR, PIECES.wQ, 0, PIECES.bB,
PIECES.bR, PIECES.bQ, 0 ];
var LoopSlideIndex = [ 0, 4];
```

KnDir dizisi bir at taşının hareket edebileceği yönleri belirtmektedir. At taşının kendi bulunduğu karenin sq değişkeni içerisinde olduğunu varsaydığımızda "i"yi de dizinin indeksi varsaydığımızda hareket edebileceği kareler; sq + KnDir[i] olmaktadır. Benzer şekilde RkDir kale taşının hareket edebileceği yönleri, BiDir fil taşının hareket edebileceği yönleri ve KiDir da şah taşının hareket edebileceği yönleri belirtmektedir. DirNum ise taşların kaç farklı yöne hareket edebileceği bilgisini tutmaktadır. Bu dizin sayesinde her bir taş için kullanılan döngü tekrar sayısı belirlenmektedir. Örneğin: PIECES.wN yani, beyaz at taşının sayısal değeri 2'dir. DirNum[PIECES.wN] yani, DirNum[2]'ın değeri

8'dir. Bu durumda bir atın 8 farklı yöne hareket edeceği belirlenmiş olmaktadır. PceDir dizini içerisine eleman olarak diğer *xx*Dir dizinleri bulunmaktadır. Böylelikle iki boyutlu bir dizi elde edilmektedir. Örneğin; öncelikli olarak PceDir[PIECES.wN] kullanılarak KnDir dizinine erişilmekte ardından da KnDir dizininin içerisinde gezinilmektedir. LoopNonSlidePce dizisi her iki renk için kayarak ilerlemeyen taşların sırayla gezilmesini sağlamaktadır. İndekslemesini ise LoopNonSlideIndex dizisi yapmaktadır. Kullanımı şu şekildedir; LoopNonSlidePce[LoopNonSlideIndex[COLOURS.WHITE]]. Bu da bizi LoopNonSlidePce dizisinin ilk indisindeki PIECES.wN taşına götürmektedir ve bir döngü ile o dizi içerisinde 0 değeriyle karşılaşılana kadar bu döngü devam ettirildiğinde PIECES.wN ve PIECES.wK gezildikten sonra döngüden çıkılmaktadır. Siyah taraf içinde aynı yapı kullanılmaktadır. Bu durumda LoopNonSlidePce[LoopNonSlideIndex[COLOURS.BLACK]] bizi dizinin üç numaralı elemanı olan PIECES.bN'a götürecektir. LoopSlidePce indeksi de benzer şekilde kayarak ilerleyen taşlar için kullanılmaktadır. 109. satırda pceIndex değeri olarak 0 veya 3 atanmaktadır. GameBoard.side = COLOURS.WHITE ise 0, COLOURS.BLACK ise 3 değeri atanmaktadır. pce değişkenine de PIECES.wN veya PIECES.bN değeri atanmakta ve daha bir sonraki taşa erişmek için pceIndex değeri bir arttırılmaktadır. 112. satırda kayarak ilerlemeyen taşları gezinmek için bir while döngüsü başlatılmaktadır. Daha önce bahsedildiği gibi bu while döngüsü 0'dan farklı olduğu sürece çalışmaktadır. Yani, ya PIECES.wN ve PIECES.wK taşlarını gezindikten sonra ya da PIECES.bN ve PIECES.bK taşlarını gezindikten sonra döngü sonlanmaktadır. 113. satırda başka bir döngü başlatılmaktadır. Bunun amacı seçilmiş olan taş çeşidinden kaç tane taş varsa hepsini gezinmektir. 114. satırda seçilmiş olan taş çeşidinin ilk taşının bulunduğu oyun karesi sq değişkenine atanmaktadır. 116. satırda ise bu taşın hareket edebileceği tüm yönlerdeki hedeflerin kontrolü için başka bir döngü başlatılmaktadır. 117. satırda ise hangi yönün kontrol edileceği dir değişkenine atanmaktadır. Gezinilen taşı PIECES.wN(2) varsaydığımızda PceDir[pce][index] dizini PceDir[2] indeksindeki KnDir dizisine erişecek ve ardından da KnDir dizisi PceDir[pce][index] dizisinde verilen index değerine göre gezilecektir. For döngüsünün ilk index değeri "0" olduğundan dolayı örneğimizde dir değişkenine "-8" değeri atanmaktadır. 118. satırda hangi oyun karesine bakılacağı sq+dir işlemi sonucu belirlenmekte ve t_sq değişkenine atanmaktadır. Örneğin; sq = 50 ise, t_sq = 50 + (-8) = 42 olacaktır. 120. satırda elde edilen t_sq değerinin oyun tahtası dışında bir alana gelip gelmediği kontrol edilmektedir. Şayet oyun tahtası dışında bir alan ise 121. satıra geçilmekte ve 121. satırdan da for döngüsünün başına dönülerek bir sonraki indisten devam edilmektedir. Şayet oyun tahtası

içerisinde kalınmışsa 124. satırdan devam edilmektedir. 124. satırda, şayet t_sq karesindeki oyun taşı karşı tarafın şahı ise bir sonraki for döngüsünün başına dönülerek bir sonraki indisten devam edilmektedir. Buradaki amaç şahın yenmesini engellemektir. Şayet bir yeme hamlesi üretilirse aynı anda hamle yapıldığından dolayı bir sonraki hamlede karşı tarafın şahı yenilerek oyun tahtası dışarısına çıkarılmış olacaktır. Bu da satranç oyun kurallarına uymamaktadır. Bu durum eş zamanlı satrançta oluşmaktadır. Çünkü klasik satrançta oluşa tehditten dolayı karşı taraf bulunduğu kareden kaçmaktadır. 124. satırdaki if yapısı false değeri dönerse 125. satırdan devam edilmektedir. Burada hamle yapılıp yapılamayacağı değerlendirilen t_sq oyun karesinin karşı tarafın tehdidi altında olup olmadığı kontrol edilmekte ve sonuç IsAttacked değişkenine atanmaktadır. 127. satırda pce değeri şah ise ve IsAttaked değişkeni BOOL.TRUE değerine sahipse hamle üretilmemekte ve bir sonraki for döngüsü indisinden devam edilmektedir. Hamle üretilmeme sebebi; şahın tehdit altındaki bir oyun karesine hamle yapmasının oyun kurallarına aykırı olmasından kaynaklanmaktadır. Burada üretilen/üretilmeyen hamleler vasıtasıyla iki şah yan yana geldikten sonra kaçış hamlesi yapıp yapamayacakları da tespit edilebilmektedir. t_sq karesinde bir taş olup olmaması durumu 131. satırda kontrol edilmekte şayet taş varsa 132. satıra geçilmektedir. 132. satırda t_sq karesindeki taşın karşı olup olmadığı kontrol edilmektedir. Şayet karşı tarafın taşıysa 133. satırdaki **AddCaptureMove** fonksiyonu çağrılarak bir yeme hamlesi kaydedilmektedir. Şayet t_sq karesi boşsa 136. satırda **AddQuietMove** fonksiyonu çağrılarak o yöne doğru yeme işlemi içermeyen bir hamle kaydedilmektedir. 138. satırda for döngüsünün başına dönülmektedir. 140. satırda LoopNonSlidePce dizisinde pceIndex'indeki oyun taşı değeri pce değişkenine atanmakta ve sonrasında pceIndex değeri bir arttırılmaktadır. 141. satırda while döngüsünün başına dönülmektedir. 143. ve 168. satırlar arasında kayarak ilerleyen taşlar (vezir, kale ve fil) için hamleler oluşturulmaktadır. 143. satırda hamle üretilecek olan tarafın indeks numarası pceIndex değişkenine atanmaktadır. 144. satırda LoopSlidePce dizisinin pceIndex'indeki oyun taşı pce değişkenine atanmaktadır. 146. satırda pce değişkeni "0" farklı olduğu sürece çalışacak bir while döngüsü başlatılmaktadır. pce "0" olması, üzerinde çalışma yapılan tarafın tüm kayarak ilerleyen taşlarının gezildiği anlamına gelmektedir. 147. satırda belirlenmiş olan taş tipindeki tüm taşları gezinmek için bir for döngüsü başlatılmaktadır. 148. satırda bu taş türünün bulunduğu karelerden ilki sq değişkenine atanmaktadır. 150. satırda belirlenmiş olan taş türünün hamle yapabileceği tüm yönleri gezinmek için bir for döngüsü başlatılmaktadır. 151. satırda belirlenmiş olan taşın hamle yapabileceği yönlerin tespit edilmesi için kullanılacak değer dir değişkenine atanmaktadır. 152. satırda

sq+dir işlemi sonucu t_sq değişkenine atanarak hamle yapılacak yön belirlenmiş olmaktadır. 154. satırda belirlenen yönde oyun tahtası dışına çıkana kadar ilerlenmesi için bir while döngüsü başlatılmaktadır. Kayarak ilerlemeyen taşlar için yazılan koddan kayarak ilerleyen taşlar için yazılan kodu ayıran kısım burasıdır. 155. satırda ilerlenen yönde karşı tarafın şahı ile karşılaşılmışsa while döngüsünden çıkılmaktadır. Çünkü karşı tarafın şahı oyun tahtasından çıkarılamayacağı için yeme hamlesi oluşturulamamaktadır. 156. satırda ilerlenen yönde bir taşla karşılaşılıp karşılaşılmadığı kontrol edilmektedir. Karşılaşılmışsa if yapısının içerisine girilmektedir. 157. satırda bu taşın karşı tarafın taşı olup olmadığı kontrol edilmektedir. Karşı tarafın taşıysa 158. satırdaki **AddCaptureMove** fonksiyonu çağırılarak bir yeme hamlesi oluşturulmaktadır. 160. satırda da while döngüsünden çıkılmaktadır. Şayet 156. satırdaki if sorgusu false dönüş yaparsa while döngüsü içerisinde 162. satıra geçilmekte ve **AddQuiteMove** fonksiyonu çağrılarak belirlenmiş olan yöne doğru bir hamle oluşturulmaktadır. 163. satırda t_sq o yöne doğru dir kadar arttırılmaktadır. Bu da bize o yönde bir sonraki karenin değerini vermektedir. while döngü şartı sağlanmadığı anda while döngüsünden çıkılmakta ve 165. satırda for döngüsünün başına dönülmektedir. for döngüsü tamamlandığında da 167. satıra geçilmektedir. 167. satırda bir sonraki taşın değeri LoopSlidePce dizisinden pce değişkenine atanmakta ve dıştaki while döngüsünün başından itibaren aynı işlemler tekrarlanmaktadır. 167. satırda aktarma işleminden sonra bir sonraki taşın indeksini elde etmek için pceIndex değeri bir arttırılmaktadır. 169. satırda da fonksiyon sonlandırılmaktadır.

3.15 Pozisyon Anahtarı Oluşturma Fonksiyonları

Oyunun başlangıcında pozisyon anahtarını oluşturmak için **GeneratePosKey** fonksiyonu çalıştırılmaktadır. Bu fonksiyon board.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function GeneratePosKey() {
2
3    var sq = 0;
4    var finalKey = 0;
5    var piece = PIECES.EMPTY;
6
7    for(sq = 0; sq < BRD_SQ_NUM; ++sq) {
8        piece = GameBoard.pieces[sq];
9        if(piece != PIECES.EMPTY && piece != SQUARES.OFFBOARD) {
```

```
10              finalKey ^= PieceKeys[(piece * 120) + sq];
11          }
12      }
13      if(GameBoard.enPas[0] != SQUARES.NO_SQ) {
14          finalKey ^= PieceKeys[GameBoard.enPas[0]];
15      }
16      if(GameBoard.enPas[1] != SQUARES.NO_SQ) {
17          finalKey ^= PieceKeys[GameBoard.enPas[1]];
18      }
19      finalKey ^= CastleKeys[GameBoard.castlePerm];
20      return finalKey;
21  }
```

GeneratePosKey fonksiyonunda 3 ve 5. satırlar arasında fonksiyonda kullanılan değişkenler tanımlanmaktadır. 7. satırda oyun tahtasındaki tüm kareleri (GameBoard.pieces) gezinecek olan bir for döngüsü başlatılmaktadır. 8. satırda GameBoard.pieces dizisinin gezilen elemanının değeri piece değişkenine atanmaktadır. piece değişkeninin değeri boş kare değilse ve oyun tahtası dışı bir değer değilse (9.satır) if yapısının içerisine girilmektedir. 10. satırda, piece değişkeni için daha önceden oluşturulmuş rastgele sayı finalKey değişkeni ile XOR işlemine tabii tutulmaktadır. Bu işlem oyun tahtasının tamamına uygulanmaktadır. Beyaz tarafın oluşturduğu bir geçerken alma karesi varsa bu karenin anahtar değeri finalKey ile XOR işlemine tabii tutulmaktadır (13 ve 15. satırlar arası). Siyah tarafın oluşturduğu geçerken alma karesi varsa bu da 16 ve 18. satırlar arasında finalKey ile XOR işlemine tabii tutulmaktadır. 19. satırda rok bilgisine ait olan anahtar değeri finalKey ile XOR işlemine tabii tutulmaktadır. 20. satırda da finalKey döndürülmektedir. Üçlü tekrarın tespitinde taraf (side) bilgisi de kullanılmaktadır ancak eş zamanlı satrançta hamle sırası her iki tarafta da olduğundan taraf bilgisi pozisyon analizindeki ayırt edici özelliğini yitirmektedir. Bu nedenle taraf bilgisine ait anahtar değeri finalKey'e eklenmemektedir.

Ayrıca oyun içerisinde taş ekleme çıkarma işlemleri sonucunda oluşacak olan yeni pozisyon anahtarları için her seferinde **GeneratePosKey** fonksiyonunu çağırmak performansı olumsuz etkileyeceğinden dolayı bu işlemleri yapmak için defs.js dosyası içerisinde kodlanmış olan aşağıdaki **HASH_PCE** fonksiyonu kullanılmaktadır;

```
1  function HASH_PCE(pce, sq) {
2      GameBoard.posKey ^= PieceKeys[(pce * 120) + sq];
3  }
```

HASH_PCE fonksiyonunda pce (oyun taşı) ve sq (oyun karesi) parametreleri verilerek fonksiyon başlatılmaktadır. 2. satırda, o oyun taşına ait olan rastgele sayı (PieceKeys[(pce * 120) + sq]) oyun tahtasının pozisyon anahtarıyla (GameBoard.posKey) XOR işlemine tabi tutulmaktadır. Şayet o karedeki oyun taşı daha önce tahtadaysa bu XOR işlemiyle oyun taşının pozisyon anahtarı oyun tahtasının pozisyon anahtarından çıkarılmaktadır. Yani bu işlem bir taş oyundan çıkarıldığında yapılmaktadır. Şayet bir taş terfi işlemi sonucu oyun tahtasına eklenmişse de aynı fonksiyon kullanılarak o taşa ait rastgele sayı oyun tahtasının pozisyon anahtarına XOR işlemi ile eklenmektedir.

Aşağıdaki **HASH_CA** fonksiyonu da rok bilgisi için benzer şekilde çalışmaktadır;

```
1  function HASH_CA() {
2        GameBoard.posKey ^= CastleKeys[GameBoard.castlePerm];
3  }
```

Geçerken alma karelerine ait rastgele sayıların, oyun tahtası pozisyon anahtar bilgisinden çıkarılması için de **HASH_EP** fonksiyonu kullanılmaktadır;

```
1  function HASH_EP() {
2     if(GameBoard.enPas[0] != SQUARES.NO_SQ)GameBoard.posKey ^=
      PieceKeys[GameBoard.enPas[0]];
3     if(GameBoard.enPas[1] != SQUARES.NO_SQ)GameBoard.posKey ^=
      PieceKeys[GameBoard.enPas[1]];
4  }
```

Eş zamanlı satranç oyununda aynı anda her iki taraf da geçerken alma karesi oluşturabildiğinden dolayı her iki tarafın da geçerken alma karesinin pozisyon anahtarı oyun tahtasının pozisyon anahtarıyla XOR işlemine tabii tutulmaktadır.

3.16 Oyun Taşının Tahtadan Çıkarılması

Bir taş yeme işlemi sonucunda bir oyun taşının tahtadan çıkarılması için MakeMove.js dosyası içerisinde aşağıdaki şekilde yer alan **ClearPiece** fonksiyonu kullanılmaktadır;

```
1  function ClearPiece(sq) {
2        var pce = GameBoard.pieces[sq];
3        var col = PieceCol[pce];
```

```
4       var index;
5       var t_pceNum = -1;
6
7       HASH_PCE(pce, sq);
8       GameBoard.pieces[sq] = PIECES.EMPTY;
9       GameBoard.material[col] -= PieceVal[pce];
10
11      for(index = 0; index < GameBoard.pceNum[pce]; ++index) {
12          if(GameBoard.pList[PCEINDEX(pce,index)] == sq) {
13              t_pceNum = index;
14              break;
15          }
16      }
17      GameBoard.pceNum[pce]--;
18      GameBoard.pList[PCEINDEX(pce, t_pceNum)] =
        GameBoard.pList[PCEINDEX(pce, GameBoard.pceNum[pce])];
19  }
```

Oyundan çıkarılacak olan taşın bulunduğu oyun karesi **ClearPiece** fonksiyonuna parametre olarak verilmektedir. 2. ve 5. satırlar arasında fonksiyonda kullanılacak olan değişkenler tanımlanmaktadır. 5. satırdaki t_pceNum değişkeni oyundan çıkarılacak olan taşın GameBoard.pList dizisindeki indeks bilgisini tutmak için kullanılmaktadır. Bir dizide hiç bir zaman "-1" indeks numarası olmayacağından dolayı da "-1" ile ilklenmektedir. 7. satırda tahtadan çıkarılacak olan taşına ait olan rastgele sayı oyun tahtasının pozisyon anahtar bilgisinden çıkarılmaktadır. 8. satırda, çıkarılmak istenen taş GameBoard.pieces dizisinden çıkarılmaktadır. 9. satırda o taşa ait materyal bilgisi GameBoard.material dizisinden çıkarılmaktadır. 11. ve 16. satırlar arasında çıkarılmak istenilen taşın GameBoard.pList dizisindeki indeks numarası tespit edilerek t_pceNum değişkenine kaydedilmektedir. 17. satırda GameBoard.pceNum dizisinde o taşın kaç tane olduğunu belirten değer bir azaltılmaktadır. 18. satırda ise GameBoard.pList dizisinde tahtadan çıkarılacak olan taşın bulunduğu yere dizideki o taşın son elemanı kaydedilmektedir. Her ne kadar GameBoard.pList dizisinde o taş tipinde bir taş hala son eleman olarak kayıtlı olsa da o taşa ait miktar bir azaltıldığı için hiçbir zaman o taş kontrol edilmeyecektir.

3.17 Oyun Taşının Tahtaya Eklenmesi

Oyun tahtasına bir taş eklemek için makemove.js dosyası içerisinde yer alan aşağıdaki **AddPiece** fonksiyonu kullanılmaktadır;

```
1  function AddPiece(sq, pce) {
2      var col = PieceCol[pce];
3      HASH_PCE(pce, sq);
4      GameBoard.pieces[sq] = pce;
5      GameBoard.material[col] += PieceVal[pce];
6      GameBoard.pList[PCEINDEX(pce, GameBoard.pceNum[pce])] = sq;
7      GameBoard.pceNum[pce]++;
8  }
```

Tahtaya eklenecek olan taş (pce) ve karesi (sq) **AddPiece** fonksiyonuna 1. satırda parametre olarak verilmektedir. 2. satırda oyun taşının rengi col değişkenine atanmaktadır. 3. satırda taşın yerleştirileceği karedeki o taşa ait rastgele sayı oyun tahtasının pozisyon anahtar değerine eklenmektedir. 4. satırda, taş GameBoard.pieces dizisinde (oyun tahtasına) parametre olarak verilen kareye eklenmektedir. 5. satırda taşın taşıdığı değer materyal bilgisine eklenmektedir. 6. satırda taş GameBoard.pList dizisinde kendine ait indeksine taşın bulunduğu kare bilgisi kaydedilmektedir. 7. satırda ise o taşa ait miktar bilgisi bir arttırılmaktadır.

3.18 Bir Taşın Hareket Ettirilmesi

Bir taşın bir kareden diğer kareye hareket ettirilmesi makemove.js dosyası içerisinde bulunan aşağıdaki **MovePiece** fonksiyonu ile yapılmaktadır;

```
1  function MovePiece(from, to){
2    var index = 0;
3    var pce = GameBoard.pieces[from];
4    HASH_PCE(pce, from);
5    GameBoard.pieces[from] = PIECES.EMPTY;
6    HASH_PCE(pce,to);
7    GameBoard.pieces[to] = pce;
8    for(index = 0; index < GameBoard.pceNum[pce]; ++index){
9      if(GameBoard.pList[PCEINDEX(pce,index)] == from) {
10       GameBoard.pList[PCEINDEX(pce,index)] = to;
11       break;
12     }
13   }
15 }
```

Taşın nereden nereye hareket ettirileceği parametre olarak 1. satırda verilmektedir. 2. satırda bir index değişkeni oluşturulmakta ve "0" ile ilklenmektedir. 3. satırda pce değişkeni tanımlanmakta ve hareket eden taş bilgisi

bu değişkene atanmaktadır. 4. satırda, bu taşın bulunduğu karedeki rastgele sayısı oyun tahtası pozisyon anahtar değerinden çıkarılmaktadır. 5. satırda taşın bulunduğu kareye boş kare bilgisi atanmaktadır. 6. satırda, taşa ait gittiği karedeki rastgele sayısı oyun tahtası pozisyon anahtar değerine eklenmektedir. 7. satırda gidilen kareye GameBoard.pieces dizisinde indeks olarak kullanılmakta ve oyun taşı buraya atanmaktadır. Böylelikle oyun tahtası üzerinde taş hareket ettirilmiş olmaktadır. 8. ve 13. kareler arasında oyun taşının bulunduğu kare from karesi yerine to karesi olarak değiştirilmektedir.

3.19 Oyun Geçmişi

Öğrenme amaçlı oyun modunda ve hamle arama motorunda hamlelerin geri alınması gerekmektedir. Bu nedenle geriye dönük olarak hangi hamlelerin yapıldığı bilgisi ve diğer pozisyon bilgilerinin bilinmesine ihtiyaç duyulmaktadır. Bunun için board.js dosyası içerisinde tanımlanan aşağıdaki değişkenler kullanılmaktadır;

```
var GameBoard{};
GameBoard.hisPly = 0;
GameBoard.history = [];
```

GameBoard.hisPly oyunda kaçıncı hamle (klasik satrançta kaçıncı yarım hamle) olduğu bilgisini tutmaktadır. GameBoard.history dizisi ise oyun boyunca yapılan hamleleri ve diğer pozisyon bilgilerini tutulmaktadır. GameBoard.history dizisi aşağıdaki **InitBoardVars** fonksiyonu kullanılarak ilklenmektedir;

```
1  function InitBoardVars() {
2    var index = 0;
3    for(index = 0; index < MAXGAMEMOVES; ++index) {
4        GameBoard.history.push( {
5            move : NOMOVE,
6            playerMove : NOMOVE,
7            opponentMove : NOMOVE,
8            castlePerm : 0,
9            wEnPas : SQUARES.NO_SQ,
10           bEnPas : SQUARES.NO_SQ,
11           fiftyMove : 0,
12           posKey : 0
13       });
14   }
```

```
15    for(index = 0; index < PVENTRIES; ++index) {
16        GameBoard.PvTable.push({
17            move : NOMOVE,
18            posKey : 0
19        });
20    }
21 }
```

InitBoardVars fonksiyonunun 2. satırında bir index değişkeni tanımlanmaktadır. 3. satırda MAXGAMEMOVES kadar (2048) tekrar yapacak bir for döngüsü başlatılmaktadır. MAXGAMEMOVES bir oyunda en fazla yapılabilecek hamle sayısını belirtmektedir. Legal bir satranç oyununda bu sayıya ulaşılması beklenmemektedir. 4. satırda, **push** fonksiyonu kullanılarak GameBoard.history dizisinin tüm elemanları ilklenmektedir. 5. satırda, arama fonksiyonu tarafından kullanılan move değişkeni ilklenmektedir. 6. satırda, oyuncuya ait hamleleri tutan playerMove değişkeni ilklenmektedir. 7. satırda, rakibin hamlelerini tutan değişken (opponentMove) ilklenmektedir. 8.satırda, rok bilgisini içeren değişken (castlePerm) ilklenmektedir. 9. satırda, beyaz tarafın geçerken alma karesine ait bilgiyi tutan değişken (wEnPas) ilklenmektedir. 10. satırda siyah tarafın geçerken alma karesine ait bilgiyi tutan değişken (bEnPas) ilklenmektedir. 11. satırda, 50 hamle kuralı bilgisini içeren değişken (fiftyMove), 12. satırda ise pozisyon anahtarı bilgisini tutan değişken (posKey) ilklenmektedir. 15. ve 20. satırlar arası en iyi pozisyonu arama işlemiyle ilgili kısımdır.

3.20 Klasik Satrançta Hamle Yapma Fonksiyonu

Klasik/sıralı satranca göre hazırlanmış olan hamle fonksiyonu aşağıdaki şekliyle makemove.js dosyası içerisinde yer almaktadır. Bu fonksiyonun eş zamanlı satranç oyununda kullanılma sebebi; en iyi pozisyonu arama fonksiyonunun klasik satranca göre olması ve bu fonksiyonun da **MakeMove** fonksiyonuna ihtiyaç duymasından kaynaklanmaktadır. Ayrıca eş zamanlı satrançta yapılan bir hamlenin geçerli hamle olup olmadığı kontrolünde yine **MakeMove** fonksiyonu kullanılmaktadır. Bu kontrolde kullanıcı tarafından belirlenen hamle **MakeMove** fonksiyonu kullanılarak gerçekleştirilmekte ve bu hamlenin kurallara uygunluğu kontrol edilmektedir. Yapılan kontrolden sonra hamle geri alınmaktadır. Eş zamanlı satranç oyununda hamlelerin yapılması için **MakeSimMove** fonksiyonu kullanılmaktadır (Bkz. 3.22).

```
1 function MakeMove(move) {
```

```
2
3   var from = FROMSQ(move);
4   var to = TOSQ(move);
5   var side = GameBoard.side;
6
7   GameBoard.history[GameBoard.hisPly].posKey = GameBoard.posKey;
8
9   if( (move & MFLAGEP) != 0) {
10      if(side == COLOURS.WHITE) {
11         ClearPiece(to-10);
12      } else {
13         ClearPiece(to+10);
14      }
15  } else if( (move & MFLAGCA) != 0) {
16      switch(to) {
17        case SQUARES.C1:
18           MovePiece(SQUARES.A1, SQUARES.D1);
19        break;
20        case SQUARES.C8:
21           MovePiece(SQUARES.A8, SQUARES.D8);
22        break;
23        case SQUARES.G1:
24           MovePiece(SQUARES.H1, SQUARES.F1);
25        break;
26        case SQUARES.G8:
27           MovePiece(SQUARES.H8, SQUARES.F8);
28        break;
29        default: break;
30      }
31  }
32
33  HASH_EP();
34  HASH_CA();
35
36  GameBoard.history[GameBoard.hisPly].move = move;
37  GameBoard.history[GameBoard.hisPly].fiftyMove =
    GameBoard.fiftyMove;
38  GameBoard.history[GameBoard.hisPly].wEnPas = GameBoard.enPas[0];
39  GameBoard.history[GameBoard.hisPly].bEnPas = GameBoard.enPas[1];
40  GameBoard.history[GameBoard.hisPly].castlePerm =
    GameBoard.castlePerm;
41
42  GameBoard.castlePerm &= CastlePerm[from];
43  GameBoard.castlePerm &= CastlePerm[to];
```

```
44
45    GameBoard.enPas[0] = SQUARES.NO_SQ;
46    GameBoard.enPas[1] = SQUARES.NO_SQ;
47
48    HASH_CA();
49
50    var captured = CAPTURED(move);
51    GameBoard.fiftyMove++;
52
53    if(captured != PIECES.EMPTY) {
54       ClearPiece(to);
55       GameBoard.fiftyMove = 0;
56    }
57
58    GameBoard.hisPly++;
59    GameBoard.ply++;
60
61    if(PiecePawn[GameBoard.pieces[from]] == BOOL.TRUE) {
62       GameBoard.fiftyMove = 0;
63       if( (move & MFLAGPS) != 0) {
64          if(side==COLOURS.WHITE) {
65             GameBoard.enPas[0]=from+10;
66             GameBoard.posKey ^= PieceKeys[GameBoard.enPas[0]];
67          } else {
68             GameBoard.enPas[1]=from-10;
69             GameBoard.posKey ^= PieceKeys[GameBoard.enPas[1]];
70          }
71       }
72    }
73
74    MovePiece(from, to);
75    var prPce = PROMOTED(move);
76    if(prPce != PIECES.EMPTY)   {
77       ClearPiece(to);
78       AddPiece(to, prPce);
79    }
80
81    GameBoard.side ^= 1;
82    if(SqAttacked(GameBoard.pList[PCEINDEX(Kings[side],0)],
      GameBoard.side))  {
83        TakeMove();
84        return BOOL.FALSE;
85    }
```

```
86      return BOOL.TRUE;
87  }
```

MakeMove fonksiyonu, move parametresi verilerek başlatılmaktadır. 3. ve 5. satırlar arasında hamlenin nereden (from) nereye (to) yapıldığını belirten ve hamle sırasının hangi renkte (side) olduğunu belirten değişkenler tanımlanmakta ve atamaları yapılmaktadır. 7. satırda, GameBoard.history dizisinde GameBoard.hisPly'ın işaret ettiği indeks numarasındaki (oyun başlangıcıysa 0. indekse) posKey değişkenine oyun tahtasının pozisyon özetleme anahtarı (GameBoard.posKey) atanmaktadır. Şayet rakibin geçerken alma karesine yapılmışsa (9. satır) hamle yapan tarafın beyaz olup olmadığı sorgulanmaktadır (10. satır). Taraf beyazsa 11. satırda to-10 karesindeki rakip piyon **ClearPiece** fonksiyonu çağrılarak oyun tahtasından çıkarılmaktadır. Şayet taraf siyahsa 13. satırda to+10 karesindeki rakip piyon oyun tahtasından çıkarılmaktadır. Bu tür bir hamleye verilen bir örnek Şekil 3-5' görülmektedir. Hamle sırası beyaz tarafta iken bir önceki hamlede siyah piyon (bP) 85. kareden 65. kareye hamle yapmıştır. Bu bir iki kare ileri başlama hamlesidir ve move değişkeni oluşturulurken MFLAGPS aktif hale getirilmektedir. Aynı zamanda 75. kare geçerken alma karesi olarak belirlenmektedir. Sıra beyaz tarafa geçtiğinde ise wP 64. kareden (from) 75. kareye (to) hamle yapmaktadır. Hamle geçerken alma karesine yapıldığı için oyun kuralı gereği o kareden geçerek iki ileri hamle yapan piyonun (bP) oyundan çıkması gerekmektedir. bP da to -10 karesinde bulunduğundan dolayı ClearPiece(to-10) şeklinde çağrılarak bP oyundan çıkarılmaktadır.

0	1	2	3	4	5	6	7	8	9	
10	11	12	13	14	15	16	17	18	19	
20	21	22	23	24	25	26	27	28	29	A1 = 21
30	31	32	33	34	35	36	37	38	39	
40	41	42	43	44	45	46	47	48	49	
50	51	52	53	54	55	56	57	58	59	
60	61	62	63	wP	bP	66	67	68	69	
70	71	72	73	74	75	76	77	78	79	
80	81	82	83	84	85	86	87	88	89	
90	91	92	93	94	95	96	97	98	99	H8 = 98
100	101	102	103	104	105	106	107	108	109	
110	111	112	113	114	115	116	117	118	119	

Şekil 3-5 Geçerken alma karesine hamle yapma

Şayet hamle geçerken alma hamlesi değilse 15. satırda hamlenin bir rok hamlesi olup olmadığı kontrol edilmektedir. Şayet hamle rok hamlesiyse 16. satırda bir switch case yapısı içerisine girilmektedir. Burada şahın rok hamlesinin ardından kalenin uygun kareye hareketi sağlanmaktadır. 33. satırda **HASH_EP** fonksiyonu kullanılarak bir önceki hamlede oyun tahtası pozisyon özetleme anahtarına eklenmiş olan geçerken alma karesinin rastgele sayısı oyun tahtası pozisyon özetleme anahtarından çıkarılmaktadır. Çünkü geçerken alma karesi bir önceki pozisyonda geçerliydi. Mevcut pozisyonda bir önceki pozisyonun geçerken alma kare bilgisi silinmektedir. 34. satırda, rok rastgele sayısı oyun tahtası pozisyon özetleme anahtarından çıkarılmakta ve 42. ve 43. satırlardaki rok bilgisi işlemlerinden sonra 48. satırda tekrar eklenmektedir. 42. ve 43. satırlarda rok bilgisi değişmez ise 48. satırda aynı anahtar tekrar eklenmektedir. 36. ve 40. satırlar arasında mevcut pozisyona ait bilgiler GameBoard.history dizisine kaydedilmektedir. 45. ve 46. satırlarda geçerken alma kareleri varsa silinmektedir. Çünkü bir önceki hamlede oluşturulan geçerken alma kareleri bir sonraki hamleden sonra geçerliliğini yitirmektedir. Yapılan hamle sonucu bir yeme işlemi oluşmuşsa yenilen taş 50. satırda captured değişkenine atanmaktadır. 51. satırda 50 hamle kuralı için tutulan değişkenin değeri bir arttırılmaktadır. İlerleyen aşamalardaki hamle analizinden piyon hareketi veya taş yeme işlemi tespit edilirse bu değişkene "0" değeri atanmaktadır. 58. ve 59. satırlarda GameBoard.hisPly ve GameBoard.ply değişkenlerinin değeri bir arttırılmaktadır. 61. satırda hamle yapan taşın piyon olup olmadığı kontrol edilmektedir. Şayet piyonsa if yapısı içerisine girilmektedir. Daha önce açıklandığı gibi GameBoard.fiftyMove değişkenine "0" değeri atanmaktadır. 63. satırda piyonun iki kare ileri hamle yapıp yapmadığı sorgulanmaktadır. Şayet yapmışsa if sorgusu içerisine girilmektedir. Burada tekrar bir if sorgusu kullanılarak siyah veya beyaz taraf için geçerken alma karesi kaydı oluşturulmaktadır. Ayrıca bu karenin pozisyon anahtar değeri olan rastgele sayı, GameBoard.posKey (oyun tahtası pozisyon özetleme anahtarı değişkeni) değişkenine eklenmektedir (XOR). 74. satırda **MovePiece** fonksiyonu kullanılarak taş from karesinden to karesine hareket ettirilmektedir. 75. satırda prPce değişkenine terfi değeri **PROMOTED** fonksiyonu çağrılarak atanmaktadır. Bir terfi işlemi varsa 76. satırdaki if yapısının içerisine girilmektedir. 77. satırda to karesindeki piyon oyundan çıkarılıp 78. satırda terfi ettiği taş oyuna dahil edilmektedir. Bu noktada hamle işlemi tamamlandığından dolayı 81. satırda, hamle yapacak olan taraf değiştirilmektedir. 82. satırda da hamle yapan tarafın şahının yapılan hamle sonucu karşı tarafın tehdidine maruz kalıp kalmadığı sorgulanmaktadır. Şayet şah tehdit altındaysa 83. satırdaki **TakeMove** fonksiyonu ile hamle geri alınmakta ve 84. satırda hamlenin yapılamayacağını belirtmek için

BOOL.FALSE değeri dönülmektedir. Şayet şah için herhangi bir tehdit yoksa 86. satırdaki BOOL.TRUE değeri döndürülmektedir.

3.21 Klasik Satrançta Hamleyi Geri Alma Fonksiyonu

Klasik sıralı satranca göre hazırlanmış olan hamleyi geri alma fonksiyonu aşağıdaki şekliyle makemove.js dosyası içerisinde **TakeMove** ismiyle yer almaktadır. **TakeMove** fonksiyonu eş zamanlı satranç oyununda iki sebepten dolayı kullanılmaktadır. Birincisi; sıralı satranca göre çalışan arama motorunda MakeMove ile yapılan hamlelerin geri alınması için kullanılmasıdır. İkincisi ise; oyuncunun hamlesinin geçerli olup olmadığının kontrolü esnasında **MakeMove** fonksiyonu ile gerçekleştirilen hamlenin **TakeMove** fonksiyonu ile geri alınmasıdır. Yapılan hamle geçerlilik kontrolünün ardından; Şayet hamle geçerli ise o hamlenin yapılmasına izin verilmektedir. Şayet hamle geçersizse o hamlenin yapılmasına izin verilmemekte ve oyuncunun yeni hamle belirlemesi beklenmektedir. Bu kontrol esnasında yapılan işlemler grafik kullanıcı ara yüzüne yansıtılmamaktadır. **TakeMove** fonksiyonu, **MakeMove** fonksiyonunun tam tersi işlem yapmaktadır.

```
 1  function TakeMove() {
 2      GameBoard.hisPly--;
 3      GameBoard.ply--;
 4
 5      var move = GameBoard.history[GameBoard.hisPly].move;
 6      var from = FROMSQ(move);
 7      var to = TOSQ(move);
 8      HASH_EP();
 9      HASH_CA();
10
11      GameBoard.castlePerm =
        GameBoard.history[GameBoard.hisPly].castlePerm;
12      GameBoard.fiftyMove =
        GameBoard.history[GameBoard.hisPly].fiftyMove;
13      GameBoard.enPas[0] = GameBoard.history[GameBoard.hisPly].wEnPas;
14      GameBoard.enPas[1] = GameBoard.history[GameBoard.hisPly].bEnPas;
15
16      HASH_EP();
17      HASH_CA();
18      GameBoard.side ^= 1;
19      if( (MFLAGEP & move) != 0) {
20          if(GameBoard.side == COLOURS.WHITE) {
```

```
21            AddPiece(to-10, PIECES.bP);
22          } else {
23            AddPiece(to+10, PIECES.wP);
24          }
25      } else if( (MFLAGCA & move) != 0) {
26          switch(to) {
27           case SQUARES.C1: MovePiece(SQUARES.D1, SQUARES.A1); break;
28           case SQUARES.C8: MovePiece(SQUARES.D8, SQUARES.A8); break;
29           case SQUARES.G1: MovePiece(SQUARES.F1, SQUARES.H1); break;
30           case SQUARES.G8: MovePiece(SQUARES.F8, SQUARES.H8); break;
31           default: break;
32          }
33      }
34
35      MovePiece(to, from);
36      var captured = CAPTURED(move);
37      if(captured != PIECES.EMPTY) {
38          AddPiece(to, captured);
39      }
40      if(PROMOTED(move) != PIECES.EMPTY)   {
41          ClearPiece(from);
42          AddPiece(from, (PieceCol[PROMOTED(move)] == COLOURS.WHITE?
            PIECES.wP : PIECES.bP));
43      }
44  }
```

TakeMove fonksiyonunun 2. satırında, hamle sayısından (GameBoard.hisPly) bir eksiltilerek n-1. tahta pozisyonunu oluşturan hamlenin indeksi elde edilmektedir. (GameBoard.hisPly değişkeni, klasik satranca ait alanlarda yarım hamle sayısını tutmakta, eş zamanlı satranca ait alanlarda ise eş zamanlı tam hamle sayısını tutmaktadır). GameBoard.hisPly değişkeni GameBoard.history dizisine indeks olarak verilerek bir önceki hamle değeri elde edilmektedir. 3. satırda da Gameboard.ply değeri bir eksiltilerek arama motoru içerisinde kullanılan bir önceki yarım hamlenin indeksi elde edilmektedir. 5. satırda, n-1. pozisyonda yapılmış olan hamle değeri move değişkenine atanmaktadır. 6. ve 7. satırlarda, bu hamlenin değişkeninin nereden nereye yapıldığı ilgili değişkenlere atanmaktadır. 8. ve 9. satırlarda, n. pozisyona ait geçerken alma ve rok rastgele sayıları, oyun tahtası pozisyon özetleme anahtar değerinden çıkarılmakta ve 16. ve 17. satırlarda n-1. pozisyondaki geçerken alma ve rok rastgele sayıları, oyun tahtası pozisyon özetleme anahtar değerine eklenmektedir. 11. ve 14. satırlar arasında n-1. oyun pozisyonuna ait değerler

ilgili değişkenlere atanmaktadır. 18. satırda oyun sırasının hangi tarafta olduğunu belirten GameBoard.side değeri 1 ise 0, 0 ise 1 yapılmaktadır. **MakeMove** fonksiyonunda bu değer benzer şekilde değiştirildiği için aynı işlem yapılarak işlem geri alınmaktadır. n-1. pozisyonda yapılan hamle bir geçerken alma hamlesi ise (19. satır) hamle yapılan karenin (to) bir gerisindeki oyun taşı tahtaya eklenmektedir (20. ve 24. satırlar arası). Şayet hamle geçerken alma hamlesi değilse hamlenin bir rok hamlesi olup olmadığı 25. satırda sorgulanmaktadır. Şayet hamle bir rok hamlesiyse 26. ve 32. satırlar arasındaki kod çalıştırılarak rok işlemi geri alınmaktadır. 35. satırda ise yapılan hamle geri alınmaktadır. 36. satırda, n-1. pozisyonda yapılan hamle sonucu yenmiş olan taş (varsa) captured değişkenine atanmaktadır. Şayet yenmiş bir taş varsa (37. satır) o taş tahtadaki eski yerine konmaktadır (38. satır). n-1. pozisyonda yapılan hamle sonucu bir terfi işlemi olmuşsa (40. satır) bu terfi ile eklenen taş tahtadan çıkarılmakta (41. satır) ve o taş terfiden önceki haliyle (piyon), hamleyi yapmadan önceki oyun karesine yerleştirilmektedir (42. satır).

3.22 Eş Zamanlı Hamle Yapma Fonksiyonu

Eş zamanlı satranç oyununun oynanabilmesi için her iki tarafın hamlesinin aynı anda ele alınması gerekmektedir. Bu işlem için makemove.js dosyası içerisinde aşağıdaki şekilde yer alan **MakeSimMove** fonksiyonu kullanılmaktadır;

```
1  function MakeSimMove(playerMove, opponentMove) {
2
3     if( (playerMove != NOMOVE) && (opponentMove != NOMOVE) &&
      (GameController.GameOver == BOOL.FALSE) ) {
4
5        var oFrom = FROMSQ(opponentMove);
6        var oTo = TOSQ(opponentMove);
7        var pFrom = FROMSQ(playerMove);
8        var pTo = TOSQ(playerMove);
9
10       var oPiece = GameBoard.pieces[oFrom];
11       var pPiece = GameBoard.pieces[pFrom];
12
13       var isUPcePawn = PiecePawn[GameBoard.pieces[pFrom]];
14       var isEPcePawn = PiecePawn[GameBoard.pieces[oFrom]];
15
16       GameBoard.history[GameBoard.hisPly].posKey = GameBoard.posKey;
17       GameBoard.history[GameBoard.hisPly].playerMove = playerMove;
```

```
18      GameBoard.history[GameBoard.hisPly].opponentMove = opponentMove;
19      GameBoard.history[GameBoard.hisPly].fiftyMove
        = GameBoard.fiftyMove;
20      GameBoard.history[GameBoard.hisPly].wEnPas = GameBoard.enPas[0];
21      GameBoard.history[GameBoard.hisPly].bEnPas = GameBoard.enPas[1];
22      GameBoard.history[GameBoard.hisPly].castlePerm
        = GameBoard.castlePerm;
23
24      if( (oTo == pTo) || (pTo == oFrom && oTo == pFrom) ){
25          if( (GameBoard.pieces[oFrom]== PIECES.bK &&
            GameBoard.pieces[pFrom]== PIECES.wP) ||
            (GameBoard.pieces[oFrom]== PIECES.wK &&
            GameBoard.pieces[pFrom]== PIECES.bP)){
26              opponentMove = opponentMove | (MOVE(oFrom, oTo,
                GameBoard.pieces[pFrom], PIECES.EMPTY, 0));
27              ClearPiece(pFrom);
28              MovePiece(oFrom, oTo);
29          }else if( (GameBoard.pieces[pFrom]== PIECES.bK &&
            GameBoard.pieces[oFrom]== PIECES.wP) ||
            (GameBoard.pieces[pFrom] == PIECES.wK &&
            GameBoard.pieces[oFrom]== PIECES.bP) ){
30              playerMove = playerMove | (MOVE(pFrom, pTo,
                GameBoard.pieces[oFrom], PIECES.EMPTY, 0));
31              ClearPiece(oFrom);
32              MovePiece(pFrom, pTo);
33          }else{
34              playerMove = playerMove | (MOVE(pFrom, pTo,
                GameBoard.pieces[oFrom], PIECES.EMPTY, 0));
35              opponentMove = opponentMove | (MOVE(oFrom, oTo,
                GameBoard.pieces[pFrom], PIECES.EMPTY, 0));
36              ClearPiece(oFrom);
37              ClearPiece(pFrom);
38          }
39
40          GameBoard.history[GameBoard.hisPly].playerMove = playerMove;
41          GameBoard.history[GameBoard.hisPly].opponentMove =
            opponentMove;
42      } else if (pTo == oFrom && oTo != pFrom){
43          if (GameBoard.pieces[oTo] != PIECES.EMPTY) ClearPiece(oTo);
44          ClearPiece(oFrom);
45          MovePiece(pFrom, pTo);
46      }else if (pTo != oFrom && oTo == pFrom){
47          if (GameBoard.pieces[pTo] != PIECES.EMPTY) ClearPiece(pTo);
48          ClearPiece(pFrom);
49          MovePiece(oFrom, oTo);
50      }else {
```

```
51          if (CAPTURED(playerMove) != PIECES.EMPTY) {
52              ClearPiece(pTo);
53          }
54          if (CAPTURED(opponentMove) != PIECES.EMPTY) {
55              ClearPiece(oTo);
56          }
57
58          MovePiece(pFrom, pTo);
59          MovePiece(oFrom, oTo);
60      }
61
62      if( (playerMove & MFLAGEP) != 0) {
63          if(playerSide == COLOURS.WHITE) {
64              if(oFrom == pTo-10){
65                  ClearPiece(oTo);
66              }
67              else{
68                  ClearPiece(pTo-10);
69              }
70          } else {
71              if(oFrom == pTo+10){
72                  ClearPiece(oTo);
73              }
74              else{
75                  ClearPiece(pTo+10);
76              }
77          }
78      }
79
80      if( (opponentMove & MFLAGEP) != 0) {
81          if(opponentSide == COLOURS.WHITE) {
82              if(pFrom == oTo-10){
83                  ClearPiece(pTo);
84              }
85              else{
86                  ClearPiece(oTo-10);
87              }
88          } else {
89              if(pFrom == oTo+10){
90                  ClearPiece(pTo);
91              }
92              else{
93                  ClearPiece(oTo+10);
```

```
94              }
95          }
96      }
97
98      if( (playerMove & MFLAGCA) != 0) {
99          switch(pTo) {
100             case SQUARES.C1:
101                 MovePiece(SQUARES.A1, SQUARES.D1);
102             break;
103             case SQUARES.C8:
104                 MovePiece(SQUARES.A8, SQUARES.D8);
105             break;
106             case SQUARES.G1:
107                 MovePiece(SQUARES.H1, SQUARES.F1);
108             break;
109             case SQUARES.G8:
110                 MovePiece(SQUARES.H8, SQUARES.F8);
111             break;
112             default: break;
113         }
114
115     }
116     if( (opponentMove & MFLAGCA) != 0) {
117         switch(oTo) {
118             case SQUARES.C1:
119               MovePiece(SQUARES.A1, SQUARES.D1);
120             break;
121             case SQUARES.C8:
122               MovePiece(SQUARES.A8, SQUARES.D8);
123             break;
124             case SQUARES.G1:
125               MovePiece(SQUARES.H1, SQUARES.F1);
126             break;
127             case SQUARES.G8:
128               MovePiece(SQUARES.H8, SQUARES.F8);
129             break;
130             default: break;
131         }
132     }
133
134     HASH_EP();
135
136     HASH_CA();
```

```
137
138         GameBoard.castlePerm &= CastlePerm[pFrom];
139         GameBoard.castlePerm &= CastlePerm[pTo];
140         GameBoard.castlePerm &= CastlePerm[oFrom];
141         GameBoard.castlePerm &= CastlePerm[oTo];
142
143         GameBoard.enPas[COLOURS.WHITE] = SQUARES.NO_SQ;
144         GameBoard.enPas[COLOURS.BLACK] = SQUARES.NO_SQ;
145
146         HASH_CA();
147
148         GameBoard.fiftyMove++;
149
150         if(CAPTURED(playerMove) != PIECES.EMPTY ||
            CAPTURED(opponentMove) != PIECES.EMPTY) {
151             GameBoard.fiftyMove = 0;
152         }
153
154         GameBoard.hisPly++;
155
156         if (isUPcePawn == BOOL.TRUE || isEPcePawn == BOOL.TRUE){
157             GameBoard.fiftyMove = 0;
158             if( (playerMove & MFLAGPS) != 0 ) {
159                 if(GameController.BoardFlipped == BOOL.FALSE){
160                     GameBoard.enPas[COLOURS.WHITE]=pFrom+10;
161                     GameBoard.posKey ^=
                        PieceKeys[GameBoard.enPas[COLOURS.WHITE]];
162                 }else if(GameController.BoardFlipped == BOOL.TRUE){
163                     GameBoard.enPas[COLOURS.BLACK]=pFrom-10;
164                     GameBoard.posKey ^=
                        PieceKeys[GameBoard.enPas[COLOURS.BLACK]];
165                 }
166             }
167             if( (opponentMove & MFLAGPS) != 0 ) {
168                 if(GameController.BoardFlipped == BOOL.FALSE){
169                     GameBoard.enPas[COLOURS.BLACK]=oFrom-10;
170                     GameBoard.posKey ^=
                        PieceKeys[GameBoard.enPas[COLOURS.BLACK]];
171                 }else if(GameController.BoardFlipped == BOOL.TRUE){
172                     GameBoard.enPas[COLOURS.WHITE]=oFrom+10;
173                     GameBoard.posKey ^=
                        PieceKeys[GameBoard.enPas[COLOURS.WHITE]];
174                 }
175             }
```

```
176         }
177         if(PROMOTED(playerMove) != PIECES.EMPTY && pTo !=
            oTo && pFrom != oTo) {
178             ClearPiece(pTo);
179             AddPiece(pTo, PROMOTED(playerMove));
180         }
181         if(PROMOTED(opponentMove) != PIECES.EMPTY && oTo !=
            pTo && oFrom != pTo) {
182             ClearPiece(oTo);
183             AddPiece(oTo, PROMOTED(opponentMove));
184         }
185
186         oMoves += (GameBoard.hisPly +"." + PceCharTr[oPiece] +
            PrMove(opponentMove)+" ");
187         pMoves += (GameBoard.hisPly +"."+ PceCharTr[pPiece] +
            PrMove(playerMove)+" ");
188
189         $("#opponentMoves").text(" + oMoves);
190         $("#playerMoves").text(" + pMoves);
191         $('textarea').scrollTop($('textarea')[0].scrollHeight);
192
193         PrintBoard();
194         SetInitialBoardPieces();
195         CheckAndSet();
196         TimerOnOpp = BOOL.TRUE;
197         TimerOn = BOOL.TRUE;
198
199         if (player == 1 & playerSide == COLOURS.WHITE){
200             GameBoard.side = COLOURS.BLACK;
201             SearchController.best = NOMOVE;
202             PreSearch();
203             if (SearchController.best != NOMOVE){
204                 TimerOnOpp = BOOL.FALSE;
205                 TotalSecondsOpp += parseInt(Increments);
206             }
207             CountDownTimerOpp.innerHTML =
                UpdateTimer(TotalSecondsOpp);
208             GameBoard.side = COLOURS.WHITE;
209         }else if (player == 1 & playerSide == COLOURS.BLACK){
210             GameBoard.side = COLOURS.WHITE;
211             SearchController.best = NOMOVE;
212             PreSearch();
213             TotalSecondsOpp += parseInt(Increments);
214             CountDownTimerOpp.innerHTML =
                UpdateTimer(TotalSecondsOpp);
```

```
215             if (SearchController.best != NOMOVE){
216                 TimerOnOpp = BOOL.FALSE;
217             }
218             GameBoard.side = COLOURS.BLACK;
219         }
220     }
221     DeselectSq(UserMove.from);
222     DeselectSq(UserMove.to);
223     $('.SqPromSelected').removeClass('SqPromSelected');
224     UserMove.from = SQUARES.NO_SQ;
225     UserMove.to = SQUARES.NO_SQ;
226     uMv = NOMOVE;
227     oppMove = NOMOVE;
228
229     if ( player == 2){
230         GameBoard.side = GameBoard.side^1;
231     }
232
233     PromotionSelectionCalled = BOOL.FALSE;
234     ProSelectDone = BOOL.FALSE;
235     moveDone = BOOL.FALSE;
236 }//end of MakeSimMove()
```

MakeSimMove fonksiyonu playerMove (oyuncunun hamlesi) ve opponentMove (rakibin hamlesi) parametrelerini alarak başlatılmaktadır. Eş zamanlı bilgisayar oyunu hem bilgisayara hem de insana karşı oynanabildiği için opponentMove bu iki rakipten hangisi seçilmişse o rakip tarafından oluşturulmakta ve gönderilmektedir. Şayet rakip bilgisayar olarak seçilmişse oyuncu ve oyuncunun bilgisayarı aynı anda düşünmeye/arama fonksiyonunu çalıştırmaya başlamaktadır. Bunun için de oyun başlar başlamaz bilgisayarın en iyi hamle arama fonksiyonu çalıştırılmaktadır. Çünkü oyuncu o sırada zaten en iyi hamlesini düşünmeye başlamıştır. Şayet karşıda başka bir insan varsa hamleler TCP/IP tabanlı bir iletişim ağı üzerinden gönderilmektedir. Herhangi bir taraf hamlesini belirlediğinde **MakeSimMove** fonksiyonu çağrılmadan önce karşı tarafın da hamlesini belirlemesi beklenilmektedir. Her iki taraf da hamlesini belirlediği anda **MakeSimMove** fonksiyonu çağrılmakta ve hamleler eş zamanlı olarak her iki taraf için de yaptırılmaktadır. **MakeSimMove** fonksiyonunun 3. satırında işlemlere başlanmadan önce her iki tarafında hamlesini belirlemiş olup olmadığına bakılmaktadır. Aynı zamanda oyunun hala devam edip etmediği de kontrol edilmektedir. Şartlar sağlanıyorsa if yapısı içerisine girilerek hamleler eş zamanlı kurallar çerçevesinde işleme alınmaktadır. 5. satırda, rakibin nereden

hamleye başladığı bilgisi oFrom değişkenine atanmaktadır. 6. satırda, rakibin nereye hamle yaptığı bilgisi oTo değişkenine atanmaktadır. 7. satırda, oyuncunun hamleye nereden başladığı pFrom değişkenine ve nereye hamle yaptığı da 8. satırda pTo değişkenine atanmaktadır. 10. satırda, rakibin hangi taşla hamle yaptığı oPiece değişkenine, 11. satırda da oyuncunun hangi taşla hamle yaptığı pPiece değişkenine atanmaktadır. Oyuncunun oynadığı taş piyon ise 13. satırda isUPcePawn değişkenine BOOL.TRUE değeri atanmaktadır. Rakibin oynadığı taş piyon ise isEPcePawn değişkenine BOOL.TRUE değeri atanmaktadır. Bu değerler elli hamle kuralı bilgisini düzenlemek için kullanılmaktadır. 16. ve 22. satırlar arasında, mevcut oyun pozisyonuna ait tüm değerler GameBoard.history dizisinde uygun alanlara atanmaktadır. Bu bilgiler mevcut oyun pozisyonunun yedeğini alarak gerektiğinde oyunu geriye almak için kullanılmaktadır. Ayrıca GameBoard.history dizisi içerisinde aynı değere sahip GameBoard.posKey'lerin olup olmadığı kontrol edilerek üçlü pozisyon tekrarı yapılıp yapılmadığı kontrol edilmektedir. 24. satırda, aynı kareye hamle (oTo == pTo) veya birbirini yeme durumu (oFrom && oTo == pFrom) olup olmadığı kontrol edilmektedir. Şayet bu durumlardan birisi mevcutsa if yapısı içerisine girilmektedir. Bu durumda öncelikli olarak kontrol edilmesi gereken husus şahın herhangi bir taşla aynı kareye hamle yapıp yapmadığıdır. Eş zamanlı satranç kuralları gereği şah sadece rakip piyonla aynı kareye hamle yapabilmektedir. Bu durumda da şah tahtadan çıkamayacağı için rakip piyon oyundan çıkarılmaktadır. 25. satırda opponentMove hamlesiyle hareket ettirilen taşın şah olup olmadığı ve karşı tarafın taşının da piyon olup olmadığı kontrolü yapılmaktadır. opponentMove hamlesini yapan taraf siyah veya beyaz olabileceği için her iki durum için de değerlendirme yapılmaktadır. Şayet 25. satırdaki if şartı sağlanıyorsa 26. satıra geçilmektedir. 26. sarıda **MOVE** fonksiyonu çağrılmakta ve karşı tarafın piyonu yenen taş (captured) parametresi olarak verilmektedir. Bu yeni hamle, eski hamle ile bitsel VEYA işlemine tabii tutulmakta ve böylelikle hem yenen taş hamleye eklenmiş olmakta hem de herhangi bir bayrak oluşmuşsa bu verinin kaybedilmesi önlenmiş olmaktadır. Kaydedilen bu veriler hem elli hareket kuralı analizinde hem de gerektiğinde hamleleri geri almak için kullanılmaktadır. 27. satırda piyon oyundan çıkarılmakta ve 28. satırda şah yeni karesine hareket ettirilmektedir. 29. ve 32. satırlar arasında, 25. ve 28. satırlar arasında opponentMove için yapılan işlemler playerMove için yapılmaktadır. Şayet hamle yapan taşlar şah ve piyon değilse 33. ve 38. satırlar arasındaki else bloğu içerisine girilmektedir. Bu alanda iki taraf da karşılıklı birbirinin taşını oyundan çıkarmaktadır. 34. satırda, playerMove'a oFrom karesindeki taş capture alanı olarak eklenmektedir. 35. satırda, opponentMove'a pFrom karesindeki taş capture alanı olarak eklenmektedir. 36. ve 37. satırlarda her

iki tarafın da hamle yapan taşı oyundan çıkarılmaktadır. Güncellenen hamle bilgileri, hamlelerin geri alınmasında kullanmak üzere GameBoard.history dizisine 40. ve 41. satırlarda kaydedilmektedir. Hamleler 24. satırdaki sorguya uymuyorsa 42. satıra atlanmaktadır. Burada oyuncunun (player), rakibin (opponent) oynadığı taşı yeme durumu ve rakibin de oyuncunun oynadığı taşı yememe durumu olup olmadığı sorgulanmaktadır. Öyleyse 43. satıra geçilmektedir. Şayet rakibin hamle yaptığı kare boş değilse yani, oyuncunun bir taşı varsa o taş (oTo karesindeki taş) oyundan çıkarılmaktadır. 44. satırda, rakibin oynadığı taş oyundan çıkarılmaktadır. Çünkü oyuncu o kareye hamle yaparak oradaki rakibin taşını yemektedir. 45. satırda da oyuncunun taşı yeni karesine hareket ettirilmektedir. 46. satırda bu sefer oyuncunun rakibin oynadığı taşı yememe ve rakibin oyuncunun oynadığı taşı yeme durumu olup olmadığı sorgulanmaktadır. Şayet öyleyse 47. satıra geçilmektedir. Burada oyuncunun hamle yaptığı karede (pTo) rakibin bir taşı varsa o taş oyundan çıkarılmaktadır. 48. satırda, oyuncunun hamle yaptığı (oynadığı) taş oyundan çıkarılmaktadır. Çünkü rakip o kareye hamle yapmaktadır. 49. satırda da rakibin oynadığı taş yeni karesine hareket ettirilmektedir. Yukarıdaki şartlardan hiçbiri sağlanmazsa 50. satırdaki else bloğu içerisine girilmektedir. Bunun anlamı; ne oyuncu, rakibin hamle yaptığı taşa ne de rakip, oyuncunun hamle yaptığı taşa doğru bir hamle yapmaktadır. Ancak hamle yapılan (oTo/pTo) yerde taş(lar) varsa o taş(lar) oyundan çıkarılmaktadır (51. ve 56. satırlar arası). 58. ve 59. satırlarda hamle yapan taşlar yeni karelerine hareket ettirilmektedir. 62. satırda oyuncunun hamlesinin (playerMove) geçerken alma karesine yapılıp yapılmadığı kontrol edilmektedir. Şayet öyleyse 63. satıra geçilmektedir. Şayet oyuncunun rengi (playerSide) beyazsa 64. satıra geçilmektedir. Şayet oyuncu geçerken alma karesine hamle yaptıysa ve bu hamleyle oyundan çıkarılması gereken rakip taş bir ileri hamle yaptıysa (oFrom == pTo-10) 65. satırda rakibin taşı hareket ettirildiği kareden silinmektedir (oTo). Kodlamanın yukarıdaki kısımlarında taşlar yeni yerlerine hareket ettirilmiş olduğu için taşlar o yeni yerlerinden silinmektedir. Şayet oyuncu geçerken alma karesine hamle yaparken bu hamle sonucu oyundan çıkarılması gereken rakip taş yerinden hareket etmemişse o taş bulunduğu yerden (pTo-10) silinmektedir (68. satır). Şayet oyuncunun rengi siyahsa 63. satırdan 70. satıra atlanmakta ve benzer işlemler 71. ve 76. satırlar arasında yapılmaktadır. Tek fark yön değiştiğinden dolayı kare numaralarının değişmesidir. 62. ve 72. satırlar arasında yapılan işlemler 80. ve 96. satırlar arasında rakip için de yapılmaktadır. 98. satırda oyuncunun hamlesinin (playerMove) bir rok hamlesi olup olmadığı sorgulanmaktadır. Şayet öyleyse 99. satırdaki switch case yapısına girilmektedir. 99. ve 113. satırlar arasındaki bu yapı içersinde pTo karesi yani, şahın gittiği kare

neresi ise o kare karşılığında hareket edilmesi gereken kareye ilgili kale hareket ettirilmektedir. Aynı işlem 116. ve 132. satırlar arasında rakibin hamlesi (opponentMove) için uygulanmaktadır. 134. satıra kadar olan kısımda geçerken alma işlemi tamamlandığı için bir önceki hamle ile oluşturulmuş olan geçerken alma karelerine ait pozisyon anahtarları (rastgele sayı) 134. satırda, geçerken alma kareleri de 143. ve 144. satırlarda silinmektedir. 136. satırda rok bilgisine ait rastgele sayı oyun tahtası pozisyon özetleme anahtarından çıkarılmaktadır. 138. ve 141. satırlar arasında rok yetkisine ait pozisyon anahtar bilgisi güncellenmektedir. Bu güncelleme defs.js dosyası içerisinde tanımlanmış olan Şekil 3-6'daki CastlePerm dizisinden faydalanılarak yapılmaktadır. Bilindiği üzeri rok bilgisi GameBoard.castlePerm değişkenine atanan tamsayının ilk dört bitinde tutulmaktadır. Örneğin; sağdaki ilk bit beyaz tarafın kısa rok (şah tarafı - WKCA) yapma yetkisi olup olmadığını göstermektedir. Bu yetki varsa en sağdaki bitin değeri "1" olmakta yoksa "0" olmaktadır. Beyaz kale hareket ettirildiğinde veya karşı taraf tarafından yenilerek oyundan çıkarıldığında kısa rok yetkisi ortadan kalkmaktadır. Bu durumda GameBoard.castlePerm & CastlePerm[28] (1111 & 1110 (onlu tabanda 14)) işlemi yapılarak yeni rok yetkisi değeri elde edilmektedir. İşlemden önce rok yetkisi 1111 ise yani, tüm roklar yapılabiliyor ise bu işlemden sonra rok yetkisi 1110 olmaktadır. Bu yeni yetki değerinde artık beyaz tarafın kısa rok yetkisi kaldırılmış olmaktadır. Beyaz şah hamle yaptığında da 1111 & 1100 işlemi yapılıp 1100 sonucu alınmaktadır. Yani sağdaki ilk iki bit "0" olduğu için beyaz tarafın artık rok yetkisi kalmamıştır. Rok yetkisini etkileyen kareler haricindeki karelerde 15 (ikili tabanda 1111) değeri olduğu için o karelerle bitsel ve (bitwise and) işlemi yapıldığında rok yetkisi değişmemektedir. 138. ve 141. satırlar arasında bu rok yetki durumu ile ilgili işlemlere ait kodlama yer almaktadır. 136. satırda oyun pozisyon özetleme anahtarından çıkarılan rok yetkisi rastgele sayısı 146. satırda oyun pozisyon özetleme anahtarına eklenmektedir. Bunun nedeni değişiklik varsa yeni rok yetkisi rastgele sayısının oyun pozisyon özetleme anahtarına eklenmesidir.

```
var CastlePerm = [
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 13, 15, 15, 15, 12, 15, 15, 14, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 7, 15, 15, 15, 3, 15, 15, 11, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15
];
```

Şekil 3-6 CastlePerm Dizisi

148. satırda elli hamle kuralı için tutulan değişkenin değeri bir arttırılmaktadır. Bu değer bir piyon hamlesi olmazsa veya bir taş yeme işlemi olmazsa arttırılmaktadır. Burada bu işlemlerin olmadığı varsayılarak arttırılmaktadır. Şayet bu işlemlerden en az birisi gerçekleşmişse devam eden bölümde bu değer sıfırlanmaktadır. Değilse zaten bir arttırıldığı için kural yerine getirilmiş olmaktadır. 150. ve 152. satırlar arasında bir taş yeme işlemi olup olmadığı sorgulanmakta ve şayet varsa elli hamle kuralı bilgisi sıfırlanmaktadır. 156. satırda hamlelerden en az birinin (oyuncu ve/veya rakip) piyon hamlesi olup olmadığı sorgulanmaktadır. Şayet öyleyse if yapısı içine girilmektedir. 157. satırda elli hamle kuralı bilgisi sıfırlanmaktadır. 158. satırda oyuncu hamlesinde MFLAGPS bayrağının (piyonun iki ileri hamlesi) işaretli olup olmadığı kontrol edilmektedir. Öyle ise 159. satırda oyuncunun tarafının beyaz olup olmadığı kontrol edilmektedir. GameController.BoardFlipped değişkeni oyuncu veya rakip için taraf beyaz ise BOOL.FALSE, siyah ise BOOL.TRUE değerini almaktadır. Taraf beyaz ise 160. satırda beyaz tarafa ait geçerken alma karesi bilgisi ilgili değişkene atanmaktadır. Bu bilgi üzerinden geçilen kare olduğu için bu kare pFrom+10 karesidir. 161. satırda, geçerken alma karesine ait pozisyon anahtar değeri (rastgele sayı) oyun pozisyon özetleme anahtar değerine eklenmektedir. 162. satırda oyuncunun tarafının siyah olup olmadığı kontrol edilmektedir. Öyleyse 163. satırda siyaha ait geçerken alma karesi bilgili ilgili değişkene atanmaktadır. 164. satırda bu karenin pozisyon anahtar değeri (rastgele sayı) oyun pozisyon özetleme anahtar değerine eklenmektedir. 158. ve 166. satırlar arasında oyuncu hamlesi için yapılan işlemlerin aynısı 167. ve 175. satırlar arasında rakip oyuncu için de yapılmaktadır. 177. satırda oyuncu hamlesi (playerMove) sonucu bir terfi işlemi olup olmadığı, oyuncu (player) ve rakibin (opponent) taşlarının aynı kareye düşüp düşmediği ve oyuncunun hamleye başladığı kareye rakibin hamle yapıp yapmadığı kontrol edilmektedir. Bu kontrollerin sebebi terfi etmeyi

hak eden taşın o esnada rakibin hamlesi sonucu tahtadan çıkıp çıkmadığını belirlemektir. Şayet terfi hak edilmiş ve oyuncunun hamle yaptığı terfi ettirilecek olan taş hala oyunda ise 178. satırda hamle yaptığı yerdeki (pTo karesindeki) taş oyundan çıkarılmakta ve yerine terfi ettiği taş 179. satırda oyuna eklenmektedir. Aynı işlemler 181. ve 184. satırlar arasında rakibin hamlesi (opponentMove) için de yapılmaktadır. 186. ve 187. satırlarda her iki tarafın yaptığı hamleler kaydedilmektedir. Örneğin; oyuncu tarafı beyaz olsun ve ilk hamlesini e2 karesinden e4 karesine yapsın. Bu durumda;

Gameboard.hisPly'de "1" değeri kayıtlıdır. pPiece'de wP yani, "1" değeri kayıtlıdır. Dolayısıyla PceCharTr[1] olmaktadır. PceCharTr dizisi defs.js dosyasında aşağıdaki şekilde tanımlanmaktadır;

```
var PceCharTr = ["","","A","F","K","V","Ş","","a","f","k","v","ş"];
```

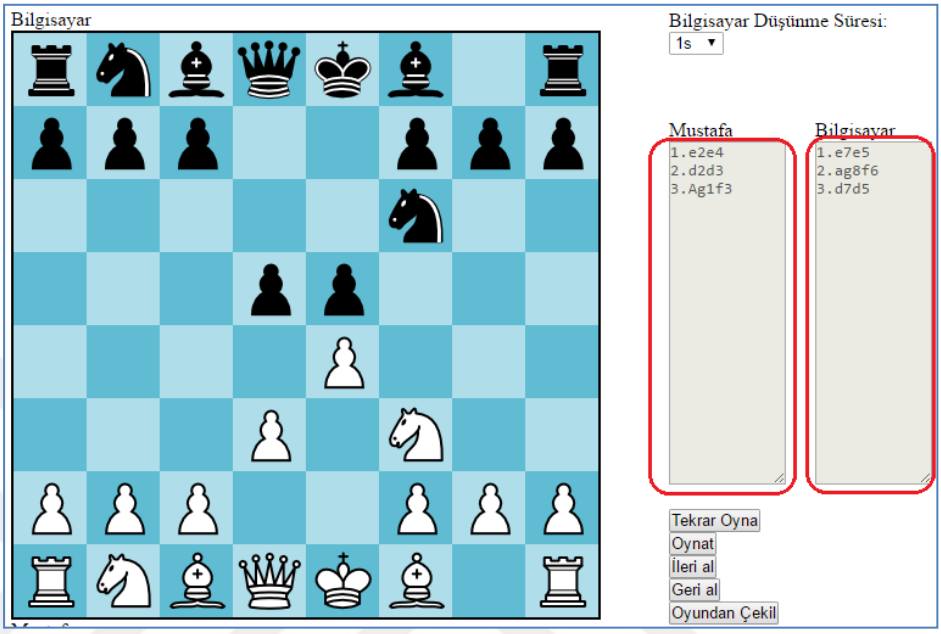
Yukarıdaki dizide görüldüğü gibi PceCharTr[1]'de kayıtlı herhangi bir değer bulunmamaktadır. Çünkü piyonların yaptığı hamlelere ait bilgiler ekrana yazdırılırken, yapılan hamle bilgisinin önüne hamleyi yapan taşın bilgisi yazdırılmamaktadır. Şayet hamle yapan taş at olsaydı bu durumda PceCharTr[2] değeri alınacaktı ve "at"ın ilk harfi olan "A" karakteri elde edilecekti. Beyaz taraf için büyük harfler, siyah taraf için ise küçük harfler kullanılmaktadır. Bu metnin oluşturulmasında son olarak io.js dosyası içerisine tanımlanmış olan aşağıdaki **PrMove** fonksiyonu kullanılmaktadır,

```
1  function PrMove(move) {
2      var MvStr;
3
4      var ff = FilesBrd[FROMSQ(move)];
5      var rf = RanksBrd[FROMSQ(move)];
6      var ft = FilesBrd[TOSQ(move)];
7      var rt = RanksBrd[TOSQ(move)];
8
9      MvStr = FileChar[ff] + RankChar[rf] + FileChar[ft] + RankChar[rt];
10
11     var promoted = PROMOTED(move);
12
13     if(promoted != PIECES.EMPTY) {
14         var pchar = 'v';
15         if(PieceKnight[promoted] == BOOL.TRUE) {
16             pchar = 'a';
17         } else if(PieceRookQueen[promoted] == BOOL.TRUE &&
```

```
            PieceBishopQueen[promoted] == BOOL.FALSE)  {
18              pchar = 'k';
19          } else if(PieceRookQueen[promoted] == BOOL.FALSE &&
            PieceBishopQueen[promoted] == BOOL.TRUE)   {
20              pchar = 'f';
21          }
22          MvStr += pchar;
23      }
24      return MvStr;
25  }
```

PrMove fonksiyonu ile bir taşın hamleye başladığı ve hamleyi bitirdiği kareler nümerik karakterlerden alfa-nümerik karakterlere dönüştürülmektedir. Örneğin; Gameboard.pieces dizisindeki 35. kareden 45. kareye yapılan hamleyi e2e4 şekline dönüştürmektedir. 1. satırda move değişkeni (yapılan hamleye ait değişken) parametre olarak alınmakta ve fonksiyon başlamaktadır. 2. satırda, bu fonksiyonun sonucunun kaydedileceği değişken tanımlanmaktadır. 4. ve 7. satırlar arasında, taşın hangi sütundan hareket ettiği ff değişkenine, hangi satırdan hareket ettiği rf değişkenine, hangi sütuna hareket ettiği ft değişkenine ve hangi satıra hareket ettiği de rt değişkenine atanmaktadır. 9. satırda, bu değişkenlerin tuttuğu değerler uygun sıra ile yan yana konularak birleştirilmekte ve MvStr değişkenine atanmaktadır. Şu ana kadar nümerik karakterlerden alfa-nümerik karakterlere dönüşüm sağlanmıştır ancak terfi yapılmışsa terfi edilen taşın ilk harfinin bu tekste eklenmesi fonksiyonun kalan kısmında yapılmaktadır. Terfi işleminin olup olmadığı 13. satırda kontrol edilmektedir. Şayet bir terfi işlemi varsa 14. satıra geçilmekte ve pchar değişkeni tanımlanarak vezir taşını temsilen "v" değeri atanmaktadır. 15. satırda terfi edilen taşın at olup olmadığı kontrol edilmekte ve öyleyse 16. satırda pchar değişkenine "a" karakteri atanmaktadır. 17. satırda terfi edilen taşın kale olup olmadığı sorgulanmakta ve öyleyse pchar'a "k" değeri atanmaktadır. 19. satırda aynı işlem fil için yapılmakta ve olumlu ise 20. satırda pchar'a "f" karakteri atanmaktadır. 22. satırda pchar karakteri MvStr'in sonuna eklenmektedir. 24. satırda, fonksiyonun çıktısı bu fonksiyonu çağıran üst fonksiyona döndürülmektedir. Sonuç olarak bir üst fonksiyon olan **MakeSimMove** fonksiyonunun 187. satırındaki pMoves değişkenine kaydedilecek olan değer "1.e2e4" şeklinde olmaktadır. Bunun anlamı da, "1. hamlede e2 karesinden e4 karesine bir piyon hareket etmiştir" olmaktadır. pMoves ve oMoves değişkenlerine kaydedilen bu hamleler oyun esnasında ekrana yazdırılmaktadır. Ekrana yazdırma işlemi de index.ejs dosyasındaki metin alanlarının içeriği değiştirilerek yapılmaktadır. Bu alanlar playerMoves ve opponentMoves id'leriyle tanımlanmış alanlardır. Bu alanlara jQuery kütüphanesi

içerisindeki bir fonksiyondan faydalanılarak 189. ve 190. satırlarında görüldüğü şekilde erişilmekte ve o kısımlara ait text alanına oMoves ve pMoves değişkenlerindeki değerler atanmaktadır. Şekil 3-7'de hamlelerin ekranda yazdırıldığı alanlar görülmektedir.



Şekil 3-7 Hamlelerin ekrana yazdırılması

191. satırda, ekrana hamlelerin yazdırıldığı alanlarda yer kalmadığında yazılar yukarı doğru kaydırılmaktadır. 193. satırda, **PrintBoard** fonksiyonu çağrılarak oyun tahtasının yeni durumu konsola yazdırılmaktadır. **SetInitialBoardPieces** fonksiyonu (Bkz. 4.4) çağrılarak oyun tahtasının yeni durumu web sayfası ekranına yazdırılmaktadır. Bu fonksiyonda tüm GameBoard.pieces dizisi gezilerek oyun taşları web sayfasındaki ilgili yerlere yerleştirilmektedir. 195. satırda, **CheckAndSet** fonksiyonu çağrılarak hamle sonrası yeni oluşan durumda oyunun bitip bitmediği kontrol edilmektedir. Oyunun bitip bitmediği kontrolü aşağıda gösterilen **CheckAndSet** fonksiyonunun 2. satırında çağrılan **CheckResult** fonksiyonu (Bkz. 3.24) ile yapılmaktadır. Şayet oyun bitmişse if yapısı içerisine girilerek GameController.GameOver değişkeni BOOL.TRUE yapılmaktadır. Bu değişken **MakeSimMove** fonksiyonunun başlangıcında kontrol edilmekte, şayet BOOL.FALSE ise yapılmış olan hamleler işleme alınmaktadır. Şayet oyun devam ediyorsa 5. satırda BOOL.FALSE değeri atanmaktadır. 6. satırda oyun bittiğinde ekrana oyunun bittiğini belirten yazının yazıldığı kısmın text alanı sıfırlanmaktadır.

```
1  function CheckAndSet() {
2        if(CheckResult() == BOOL.TRUE) {
3              GameController.GameOver = BOOL.TRUE;
4        } else {
5              GameController.GameOver = BOOL.FALSE;
6              $("#GameStatus").text('');
7        }
8  }
```

MakeSimMove fonksiyonunun 196. ve 197. satırlarında, TimerOn ve TimerOnOpp değişkenlerinin değerleri BOOL.TRUE yapılmaktadır. Bu değişkenler defs.js içerisinde tanımlıdır ve her iki taraf için tutulan sayaçların kontrolünde kullanılmaktadır. TimerOn oyuncu için tutulan sayacı, TimerOnOpp ise rakip için tutulan sayacı kontrol etmektedir. Bir taraf hamlesini yaptığında karşı taraf hala hamle yapmamışsa hamle yapan tarafın sayacı durdurulmaktadır. TimerOn veya TimerOnOpp değişkenlerine atanan değer BOOL.FALSE ise sayaç durdurulmakta BOOL.TRUE ise sayaç çalıştırılmaktadır (Bkz. 4.3). 196. ve 197. satırlarda bu değişken değerlerinin BOOL.TRUE yapılma sebebi; mevcut durumda her iki tarafın yaptığı hamlelerin oyun tahtası üzerine yansıtılmış olması yani, hamleler ile ilgili işlemlerin tamamlanmış olması ve yeni hamle için beklenmeye başlanmış olmasıdır. Bu nedenle bu bölümün devam eden kısımlarında yeni hamle ile ilgili işlemler ele alınmaktadır. Bilindiği üzere eş zamanlı satranç oyununda rakip bilgisayar veya insan olabilmektedir. Bu fonksiyon içerisinde rakibin bilgisayar olduğu durum ele alınmaktadır. Rakibin insan olduğu durumda kodlama yapısı farklı olduğu için (bir ağ iletişimi gerektirdiği için) diğer benzer kodlamalarla birlikte index.js dosyası içerinde ele alınmaktadır. 199. satırda oyunun bilgisayara karşı oynanıp oynanmadığı (player ==1) ve oyuncunun tarafının beyaz olup olmadığı kontrol edilmektedir. Şartlar uyuyorsa if yapısı içerisine girilmektedir. Oyuncu tarafı beyaz olduğundan dolayı bilgisayar tarafı da siyah olmaktadır ve bilgisayarın en iyi hamle aramasını yapabilmesi için oyun sırasının bilgisayarda olması gerekmektedir. Her ne kadar eş zamanlı satrançta bir oyun sırası yoksa da bilgisayara hangi taraf için arama yapacağının bildirilmesi gerekmektedir (200. satır). 201. satırda en iyi hamleyi tutan değişkene NOMOVE değeri atanmaktadır. Bunun nedeni şayet 202. satırda çağrılan **PreSearch** fonksiyonu ile bir hamle bulunamazsa bir önceki aramada en iyi hamle olarak atanan değerin en iyi hamle olarak kalmasını engellemektir. Bu engellemenin nedeni; şayet bilgisayar herhangi bir hamle bulamazsa yani, oyunu bitirecek bir durum oluştuğunda sayacının durmasını engellemektir. Bu durum 203. satırdaki sorgu ile kontrol edilmektedir. Şayet bilgisayar geçerli bir hamle

bulmuşsa 204. satırda bilgisayarın sayacı durdurulmaktadır. 205. Satırda da her geçerli hamle için eklenen artış miktarı bilgisayarın sayacına ilave edilmektedir. Bu durum oyun arttırmalı oynandığında etkili olmaktadır. Oyun arttırmalı oynanmıyorsa Increments değeri "0" olacağından dolayı sayaç değeri değişmeyecektir. 207. satırda rakibe (bu durumda bilgisayar) ait sayaç değeri güncellenmektedir. Oyuncunun hamle yapabilmesi için 208. satırda oyun tarafı beyaz olarak değiştirilmektedir. 199. ve 208. satırlar arasında yapılan işlemlerin aynısı, oyuncu tarafı siyah taraf olduğunda 209. ve 219. satırlar arasında yapılmaktadır. 221. ve 222. satırlarda hamle yapmak için kullanıcı tarafından işaretlenmiş olan karelerin işaretleri **DeselectSq** fonksiyonu (Bkz. 4.5) kullanılarak kaldırılmaktadır. 223. satırda ise terfi edecek olan taşın bulunduğu karedeki koyulaştırma (high-ligth) silinmektedir. Başka bir deyişle; terfi edilecek taşların listesi üzerinde yapılan seçim işareti silinmektedir. 224. ve 225. karelerde kullanıcının hamleyi nereden nereye yaptığını gösteren karelere ait bilgiler sıfırlanmaktadır. Çünkü yeni hamle yapılırken buradaki değişkenlerin boş mu dolu mu olduğu kontrolleri yapılmaktadır. 226. ve 227. satırlarda oyuncu ve rakibinin hamlelerine ait bilgiler yeni hamlelerin kaydedilmesi için sıfırlanmaktadır. 229. ve 231. satırlar arasında şayet ağ üzerinden iki oyuncu karşılıklı oynuyorsa oyun sırası tarafı tersine çevrilerek oyun sırası oyuncuya verilmektedir (**CheckResult** fonksiyonu tarafından bazı kontrollerin yapılması için oyun tarafı tersine çevriliyor. player == 1 durumu için düzenleme diğer alanlarda yapılmaktadır.). Böylelikle **GenerateMoves** fonksiyonu tarafından oyuncu için doğru hamlelerin üretilmesi sağlanmaktadır. Hamleye ait işlemler bittiği için kullanılan bayraklara 233. ve 235. satırlar arasında BOOL.FALSE değeri atanmaktadır.

3.23 Eş Zamanlı Satrançta Hamleyi Geri Alma Fonksiyonu

Eş zamanlı satranç oyununda yapılan bir hamlenin geri alınması için makemove.js dosyası içerisinde yer alan **TakeSimMove** fonksiyonu kullanılmaktadır;

```
1  function TakeSimMove(){
2    GameBoard.hisPly--;
3
4    var playerMove = GameBoard.history[GameBoard.hisPly].playerMove;
5    var opponentMove = GameBoard.history[GameBoard.hisPly].opponentMove;
6    var oFrom = FROMSQ(opponentMove);
7    var oTo = TOSQ(opponentMove);
```

```
8   var pFrom = FROMSQ(playerMove);
9   var pTo = TOSQ(playerMove);
10
11  HASH_EP();
12  HASH_CA();
13
14  GameBoard.castlePerm = GameBoard.history[GameBoard.hisPly].
    castlePerm;
15  GameBoard.fiftyMove = GameBoard.history[GameBoard.hisPly].fiftyMove;
16  GameBoard.enPas[0] = GameBoard.history[GameBoard.hisPly].wEnPas;
17  GameBoard.enPas[1] = GameBoard.history[GameBoard.hisPly].bEnPas;
18
19  HASH_EP();
20  HASH_CA();
21
22  if( (oTo == pTo) || (pTo == oFrom && oTo == pFrom) ){
23    if( (GameBoard.pieces[oTo]== PIECES.bK && CAPTURED
      (opponentMove) == PIECES.wP) || (GameBoard.pieces[oTo]==
      PIECES.wK && CAPTURED(opponentMove)== PIECES.bP)){
24      console.log( " oTo: "+ oTo + " oFrom: "+oFrom+"
        CAPTURE(opponentMove): " +CAPTURED(opponentMove) );
25      console.log( " pTo: "+ pTo + " pFrom: "+pFrom+"
        CAPTURE(playerMove): " +CAPTURED(playerMove) );
26      MovePiece(oTo, oFrom);
27      AddPiece(pFrom, CAPTURED(opponentMove));
28    }else if( (GameBoard.pieces[pTo]== PIECES.bK &&
      CAPTURED(playerMove) == PIECES.wP) || (GameBoard.pieces[pTo]
      == PIECES.wK && CAPTURED(playerMove)== PIECES.bP ) ){
29      console.log( " oTo: "+ oTo + " oFrom: "+oFrom+"
        CAPTURE(opponentMove): " +CAPTURED(opponentMove) );
30      console.log( " pTo: "+ pTo + " pFrom: "+pFrom+"
        CAPTURE(playerMove): " +CAPTURED(playerMove) );
31      MovePiece(pTo, pFrom);
32      AddPiece(oFrom, CAPTURED(playerMove));
33    }else{
34       AddPiece(pFrom, CAPTURED(opponentMove));
35       AddPiece(oFrom, CAPTURED(playerMove));
36    }
37  } else if (pTo == oFrom && oTo != pFrom){
38      MovePiece(pTo, pFrom);
39      AddPiece(pTo, CAPTURED(playerMove));
40      if ( CAPTURED(opponentMove) != PIECES.EMPTY)
        AddPiece(oTo, CAPTURED(opponentMove));
41  }else if (pTo != oFrom && oTo == pFrom){
42    MovePiece(oTo, oFrom);
```

```
43      AddPiece(oTo, CAPTURED(opponentMove));
44      if ( CAPTURED(playerMove) != PIECES.EMPTY)
        AddPiece(pTo, CAPTURED(playerMove));
45    }else {
46        if (CAPTURED(playerMove) != PIECES.EMPTY) {
47            MovePiece(pTo, pFrom);
48            AddPiece(pTo, CAPTURED(playerMove));
49        }else{
50            MovePiece(pTo, pFrom);
51        }
52        if (CAPTURED(opponentMove) != PIECES.EMPTY) {
53            MovePiece(oTo, oFrom);
54            AddPiece(oTo, CAPTURED(opponentMove));
55        }else{
56            MovePiece(oTo, oFrom);
57        }
58    }
59
60    if( (playerMove & MFLAGEP) != 0) {
61        if(playerSide == COLOURS.WHITE) {
62            AddPiece(pTo-10, PIECES.bP);
63        } else {
64            AddPiece(pTo+10, PIECES.wP);
65        }
66    }
67    if( (opponentMove & MFLAGEP) != 0) {
68        if(opponentSide == COLOURS.WHITE) {
69            AddPiece(oTo-10, PIECES.bP);
70        } else {
71            AddPiece(oTo+10, PIECES.wP);
72        }
73    }
74
75    if( (MFLAGCA & playerMove) != 0) {
76      switch(pTo) {
77        case SQUARES.C1: MovePiece(SQUARES.D1, SQUARES.A1); break;
78        case SQUARES.C8: MovePiece(SQUARES.D8, SQUARES.A8); break;
79        case SQUARES.G1: MovePiece(SQUARES.F1, SQUARES.H1); break;
80        case SQUARES.G8: MovePiece(SQUARES.F8, SQUARES.H8); break;
81        default: break;
82       }
83    }
84    if( (MFLAGCA & opponentMove) != 0) {
```

```
85      switch(oTo) {
86          case SQUARES.C1: MovePiece(SQUARES.D1, SQUARES.A1); break;
87          case SQUARES.C8: MovePiece(SQUARES.D8, SQUARES.A8); break;
88          case SQUARES.G1: MovePiece(SQUARES.F1, SQUARES.H1); break;
89          case SQUARES.G8: MovePiece(SQUARES.F8, SQUARES.H8); break;
90          default: break;
91      }
92  }
93
94  if(PROMOTED(playerMove) != PIECES.EMPTY){
95      ClearPiece(pFrom);
96      AddPiece(pFrom, (PieceCol[PROMOTED(playerMove)] ==
        COLOURS.WHITE ? PIECES.wP : PIECES.bP));
97  }
98  if(PROMOTED(opponentMove) != PIECES.EMPTY){
99      ClearPiece(oFrom);
100     AddPiece(oFrom, (PieceCol[PROMOTED(opponentMove)] ==
        COLOURS.WHITE ? PIECES.wP : PIECES.bP));
101 }
102
103 PrintBoard();
104 SetInitialBoardPieces();
105
106 if(PROMOTED(opponentMove)) oMoves =
    oMoves.substr(0,oMoves.length-1);
107 if(PROMOTED(playerMove)) pMoves = pMoves.substr(0,pMoves.length-1);
108 var opponentMoveSubNumber = 10;
109 var playerMoveSubNumber = 10;
110 if(GameBoard.pieces[oFrom] == PIECES.wP || GameBoard.pieces[oFrom]
    == PIECES.bP){
111     opponentMoveSubNumber = 9;
112 }
113 if(GameBoard.pieces[pFrom] == PIECES.wP || GameBoard.pieces[pFrom]
    == PIECES.bP){
114     playerMoveSubNumber = 9;
115 }
116
117 oMoves = oMoves.substr( 0, (oMoves.length - (GameBoard.hisPly >= 9 ?
    (GameBoard.hisPly >= 99?opponentMoveSubNumber:
    opponentMoveSubNumber-1) : opponentMoveSubNumber-2)) );
118 pMoves = pMoves.substr( 0, (pMoves.length - (GameBoard.hisPly >= 9 ?
    (GameBoard.hisPly >= 99?playerMoveSubNumber:
    playerMoveSubNumber-1) : playerMoveSubNumber-2)) );
119 $("#opponentMoves").text(" + oMoves);
120 $("#playerMoves").text(" + pMoves);
```

```
121
122    if(GameController.GameOver == BOOL.TRUE){
123        GameController.GameOver = BOOL.FALSE;
124        $("#GameStatus").text('');
125    }
126    SearchController.best = opponentMove;
127  }//End of TakeSimMove()
```

TakeSimMove fonksiyonunun 2. satırında GameBoard.hisply değeri bir azaltılarak n-1. oyun pozisyonunda yapılmış olan hamlenin GameBoard.history dizisindeki indeksi elde edilmektedir. 4. ve 5. satırlarda, n-1. oyun pozisyonunda oyuncu ve rakip tarafından yapılan hamleler ilgili değişkenlere atanmaktadır. 6. ve 7. satırlarda, rakibin nereden nereye hamle yaptığı ilgili değişkenlere atanmaktadır. 8. ve 9. satırlarda, oyuncunun nereden nereye hamle yaptığı ilgili değişkenlere atanmaktadır. 11. ve 12. satırlarda, n. oyun pozisyonundaki geçerken alma ve rok rastgele sayıları, oyun tahtası pozisyon özetleme anahtar değerinden çıkarılmaktadır. 14. ve 17. satırlarda, n-1. oyun pozisyonuna ait bilgiler ilgili değişkenlere atanmaktadır. 19. ve .20. satırlarda, n-1. oyun pozisyonuna ait geçerken ve rok rastgele sayıları oyun anahtar özetleme değerine eklenmektedir. Oyuncuların aynı kareye hamle yapması veya birbirini yemesi durumunda (22. satır) 23. ve 36. satırlar arasındaki kod çalıştırılmaktadır. n-1. oyun pozisyonunda, rakibin hamle yaptığı taş şah ise ve oyuncunun hamle yaptığı taş da piyonsa (23. satır) rakibin taşı geriye doğru hareket ettirilmekte (26. satır) ve oyundan çıkarılmış olan oyuncunun taşı da tahtaya yeniden eklenmektedir (27. satır). Şayet n-1. oyun pozisyonunda oyuncunun hamle yaptığı taş şah ve rakibin hamle yaptığı taş piyonsa (28. satır) aynı işlemler taraflar değiştirilerek yapılmaktadır (29. ve 32. satırlar). Şayet hamle yapılan taşlardan birisi şah birisi de piyon değilse 33. satırdaki "else" ile başlayan alana geçilmektedir. 34. ve 35. satırda, birbirini yeme veya aynı kareye hamle yapma sonucu oyundan çıkarılmış olan taşlar n-1. oyun pozisyonundaki yerlerine (pFrom/oFrom) konulmaktadır. Şayet n-1. oyun pozisyonunda yapılan hamlede oyuncu, rakibin hamle yapan taşını yemiş ve rakibin hamle yapan taşı, oyuncunun hamle yapan taşını yememişse (37. satır) oyuncunun taşı geriye doğru hareket ettirilerek n-1. pozisyondaki yerine taşınmaktadır (38. satır) ve rakibin taşı oyun tahtasına eklenerek n-1. pozisyondaki yerine konulmaktadır (39. satır). Şayet rakip, n-1. pozisyonda yaptığı hamleyle oyuncunun bir taşını yemişse oyuncunun taşı n-1. pozisyondaki yerine konulmaktadır (40. satır). Şayet rakip, oyuncunun hamle yapan taşını yemiş ve oyuncu, rakibin hamle yapan taşını yememişse (41. satır) rakibin taşı n-1. pozisyondaki yerine taşınmakta ve oyuncunun taşı da tahtaya eklenerek n-1.

pozisyondaki yerine konulmaktadır (42. ve 43. satırlar). Şayet oyuncu rakibin bir taşını yemişse o taş n-1. pozisyondaki yerine konulmaktadır (44. satır). Şayet 22. satırdan itibaren sorgulanan şartlardan hiçbiri gerçekleşmemişse yani, her iki oyuncu da birbirinin hamle yapan taşını yememişse (45. satır) 46. ve 58. satırlar arasındaki kod çalıştırılmaktadır. Şayet oyuncu rakibin bir taşını yemişse (46. satır) oyuncunun taşı n-1. pozisyondaki yerine taşınmakta ve rakibin taşı n-1. pozisyondaki yerine eklenmektedir (47. ve 48. satırlar). Şayet oyuncu rakibin bir taşını yememişse oyuncunun taşı n-1. pozisyondaki yerine taşınmaktadır. 46. ve 51. satırlar arasında oyuncu için yapılan işlemler 52. ve 57. satırlar arasında rakip için de yapılmaktadır. Şayet oyuncu geçerken alma hamlesiyle rakibin taşını yemişse (60. satır) rakibin taşı eski yerine (n-1. pozisyon) konulmaktadır (61 ve 65. satırlar). Şayet rakip geçerken alma hamlesiyle oyuncunun taşını yemişse (67. satır) oyuncunun taşı eski yerine konulmaktadır (68. ve 72. satırlar). Şayet oyuncunun hamlesi bir rok hamlesi ise (75. satır) rok hamlesinin tamamlanması için gerekli olan hamle 76. ve 82. satırlar arasında geri alınmaktadır. Benzer şekilde rakip bir rok hamlesi yapmışsa (84. satır) 85. ve 91. satırlar arasında rakibin rok hamlesi geri alınmaktadır. Şayet oyuncunun yaptığı hamlede bir terfi işlemi varsa (94. satır) terfi etmiş olan taş oyundan çıkarılmakta (95. satır) ve o taşın terfi etmeden önceki hali (piyon) n-1. pozisyondaki yerine konulmaktadır (96. satır). Aynı işlem rakip için 98. ve 101. satırlar arasında yapılmaktadır. 103. satırda, oyun tahtası konsola yazdırılmaktadır. 104. satırda, oyun tahtası grafik ara yüze yazdırılmaktadır. 106. ve 120. satırlar arasında, yapılan hamlelere ait bilgiler ile ilgili işlemler yapılmaktadır. Şayet hamlede bir terfi işlemi varsa hangi taşa terfi edilmişse o taşa ait harf hamleyle ilgili tutulan metinin sonuna eklenmektedir. 106. ve 107. satırlarda, bu harf silinerek metin standart bir hale getirilmektedir. 108. ve 109. satırlarda, bir hamleye ait tutulan bilginin metin uzunluğu belirlenmektedir. Piyon haricindeki diğer taş hamlelerinde taşın baş harfi metine eklendiği için metinlerden o taşa ait hamle bilgisini çıkarırken onlardan 10, piyonlardan ise 9 karakter çıkarılmaktadır. Şayet n-1. pozisyonda yapılan hamle bir piyon hamlesiyse belirlenen değişkene "9" değeri atanmaktadır (110. ve 115. satırlar). 117. ve 118. satırlarda, n-1. pozisyonda yapılan hamleye ait bilgi o bilgiyi tutan değişkenden çıkarılmaktadır. 119. ve 120. satırlarda, hamlelere ait bilgi n-1. pozisyondaki haliyle ekrana yazdırılmaktadır. Şayet oyun sona ermişse (122. satır) oyun devam ettirilmekte (123. satır) ve oyunun bittiğini gösteren yazı ekrandan kaldırılmaktadır (124. satır). 126. satırda, yeni seçilmiş en iyi hamle yerine history'deki en iyi hamle atanmaktadır.

3.24 Yapılan Eş Zamanlı Hamle Sonucunda Oluşan Pozisyonun İncelenmesi

Yapılan eş zamanlı hamle sonucunda oluşan pozisyon incelemesi, makemove.js dosyası içerisinde aşağıdaki şekilde yer alan **CheckResult** fonksiyonu kullanılarak yapılmaktadır;

```
1  function CheckResult() {
2    if(GameBoard.fiftyMove >= 50) {
3      $('#NewGameButton').attr("disabled", false);
4      $("#GameStatus").text("OYUN BERABERE{50 hareket kuralı}");
5      return BOOL.TRUE;
6    }
7
8    if (ThreeFoldRep() >= 2) {
9      $('#NewGameButton').attr("disabled", false);
10     $("#GameStatus").text("OYUN BERABERE{Üçlü tekrarlama kuralı}");
11     return BOOL.TRUE;
12   }
13
14   if (DrawMaterial() == BOOL.TRUE) {
15     $('#NewGameButton').attr("disabled", false);
16     $("#GameStatus").text("OYUN BERABERE
       {Mat yapmak için yetersiz materyal}");
17     return BOOL.TRUE;
18   }
19
20   MoveFound = [0,0];
21   AnyMoveExist();
22   GameBoard.side = GameBoard.side^1;
23   AnyMoveExist();
24   if( (MoveFound[0] != 0) && (MoveFound[1] != 0)) return BOOL.FALSE;
25
26   var SideInCheck = SqAttacked(GameBoard.pList[PCEINDEX
     (Kings[GameBoard.side],0)], GameBoard.side^1);
27   var OpSideInCheck = SqAttacked(GameBoard.pList[PCEINDEX
     (Kings[GameBoard.side^1],0)], GameBoard.side);
28
29   if ( (SideInCheck == BOOL.TRUE) &&
     (MoveFound[GameBoard.side]==0) &&
     (OpSideInCheck == BOOL.TRUE) &&
     (MoveFound[GameBoard.side^1] == 0) ){
30     $('#NewGameButton').attr("disabled", false);
```

```
31          $("#GameStatus").text("OYUN BERABERE {Karşılıklı mat}");
32          return BOOL.TRUE;
33      }else if( (SideInCheck == BOOL.TRUE) &&
        (MoveFound[GameBoard.side] == 0) ){
34          $('#NewGameButton').attr("disabled", false);
35          if(GameBoard.side == COLOURS.WHITE) {
36          $("#GameStatus").text("OYUN BİTTİ {Siyah Mat etti}");
37          return BOOL.TRUE;
38          }else {
39              $("#GameStatus").text("OYUN BİTTİ {Beyaz mat etti}");
40              return BOOL.TRUE;
41          }
42      } else if( (OpSideInCheck == BOOL.TRUE) &&
        (MoveFound[GameBoard.side^1] == 0) ){
43          $('#NewGameButton').attr("disabled", false);
44          if(GameBoard.side^1 == COLOURS.WHITE) {
45              $("#GameStatus").text("OYUN BİTTİ {Siyah Mat etti}");
46              return BOOL.TRUE;
47          } else {
48              $("#GameStatus").text("OYUN BİTTİ {Beyaz mat etti}");
49              return BOOL.TRUE;
50          }
51      } else if( (MoveFound[0] == 0) || (MoveFound[1] == 0)) {
52          $('#NewGameButton').attr("disabled", false);
53          $("#GameStatus").text("OYUN BERABERE {Çıkmaz}");
54          return BOOL.TRUE;
55      }
56      return BOOL.FALSE;
57  }
```

CheckResult fonksiyonunun 2. ve 6. satırları arasında, elli hamle kuralının gerçekleşip geçekleşmediği kontrol edilmekte ve gerçekleşmişse oyunun berabere sonuçlandığı ekrana yazdırılmakta ve oyunun sonlanması için BOOL.TRUE değeri döndürülmektedir. 8. ve 12. satırlar arasında, üçlü tekrar kuralının gerçekleşip gerçekleşmediği kontrol edilmektedir. Şayet bu kural gerçekleşmişse oyunun berabere sonuçlandığı ekrana yazdırılmakta ve oyunun sonlanması için BOOL.TRUE değeri döndürülmektedir. Üçlü tekrar kuralının gerçekleşip gerçekleşmediğinin tespitinde kullanılan **ThreeFoldRep** fonksiyonu aşağıdaki gibidir;

```
1  function ThreeFoldRep() {
2      var i = 0, r = 0;
```

```
3
4     for(i = 0; i < GameBoard.hisPly; ++i) {
5         if (GameBoard.history[i].posKey == GameBoard.posKey) {
6             r++;
7         }
8     }
9     return r;
10 }
```

ThreeFoldRep fonksiyonu ile mevcut pozisyona ait pozisyon özetleme anahtarının geçmişte tekrar edip etmediği kontrol edilmekte ve tekrar etmişse kaç defa tekrar ettiği kontrol edilerek döndürülmektedir.

CheckResult fonksiyonunun 14. ve 18. satırları arasında, mat yapmak için yeterli eleman olup olmadığı kontrol edilmekte ve öyleyse oyunun sonlandığı ekrana yazdırılmakta ve oyunun sonlandırılması için BOOL.TRUE değeri döndürülmektedir. Burada kullanılan **DrawMaterial** fonksiyonu makemove.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function DrawMaterial() {
2      if (GameBoard.pceNum[PIECES.wP]!=0 ||
       GameBoard.pceNum[PIECES.bP]!=0) return BOOL.FALSE;
3      if (GameBoard.pceNum[PIECES.wQ]!=0 ||
       GameBoard.pceNum[PIECES.bQ]!=0 ||
       GameBoard.pceNum[PIECES.wR]!=0 ||
       GameBoard.pceNum[PIECES.bR]!=0) return BOOL.FALSE;
4      if (GameBoard.pceNum[PIECES.wB] > 1 ||
       GameBoard.pceNum[PIECES.bB] > 1) {return BOOL.FALSE;}
5      if (GameBoard.pceNum[PIECES.wN] > 1 ||
       GameBoard.pceNum[PIECES.bN] > 1) {return BOOL.FALSE;}
6      if (GameBoard.pceNum[PIECES.wN]!=0 &&
       GameBoard.pceNum[PIECES.wB]!=0) {return BOOL.FALSE;}
7      if (GameBoard.pceNum[PIECES.bN]!=0 &&
       GameBoard.pceNum[PIECES.bB]!=0) {return BOOL.FALSE;}
8      return BOOL.TRUE;
9  }
```

DrawMaterial fonksiyonu içerisinde yukarıdaki şekilde bazı taş çeşitlerinden mat yapmaya yetecek kadar taş bulunuyorsa BOOL.FALSE değeri döndürülmektedir. Mat yapacak kadar taş bulunmuyorsa BOOL.TRUE değeri döndürülmektedir. Bu fonksiyon ile her iki tarafın da taşları kontrol edildiği için

her iki tarafın birden mat edecek kadar taşı bulunmazsa BOOL.TRUE değeri döndürülmektedir.

CheckResult fonksiyonunda MoveFound isimli bir dizi oluşturulmakta ve bu dizi 21. satırda çağrılan **AnyMoveExist** fonksiyonunda kullanılmaktadır. **AnyMoveExist** fonksiyonu ile tarafların geçerli hamleleri olup olmadığı kontrol edilmektedir.

```
1  function AnyMoveExist(){
2      GenerateMoves();
3      var index = 0;
4      for( index = GameBoard.moveListStart[GameBoard.ply];
       index < GameBoard.moveListStart[GameBoard.ply + 1]; ++index )  {
5          if ( MakeMove(GameBoard.moveList[index]) == BOOL.FALSE)  {
6              continue;
7          }
8          TakeMove();
9          MoveFound[GameBoard.side]++;
10         break;
11     }
12 }
```

AnyMoveExist fonksiyonunda, GameBoard.side hangi tarafı gösteriyorsa o tarafın hamleleri üretilmekte ve herhangi bir hamle varsa MoveFound dizisi içerisinde o tarafa ait hamle miktarı bir arttırılarak fonksiyondan çıkılmaktadır.

CheckResult fonksiyonun 22. satırında GameBoard.side değişkeninde taraf değiştirilmekte ve 23. satırda o tarafın geçerli hamlesi olup olmadığı kontrol edilmektedir. Şayet her iki tarafında geçerli hamlesi varsa BOOL.FALSE değeri dönülerek oyunun devam etmesi sağlanmaktadır (24. satır). 26. ve 27. satırlarda her iki tarafa da şah çekilip çekilmediği kontrol edilerek sonuç ilgili değişkenlere atanmaktadır. 29. ve 55. satırlar arasında da oyunu berabere sonuçlandıracak veya bir tarafı mat edecek durumun oluşup oluşmadığı if else yapısı yardımıyla kontrol edilmekte ve bu durumlardan birisi oluşuyorsa oyunun sonlanması için BOOL.TRUE değeri döndürülmektedir. Şayet oyunu sonlandıracak bir durum oluşmamışsa BOOL.FALSE değeri döndürülmektedir (56. satır).

3.25 Oyun Süresinin Tutulması (Sayaç)

Oyunun belirli bir süre içerisinde oynanması için oyun kurulumu esnasında bir süre belirlenebilmektedir. Yine oyun kurulumu esnasında bu sürenin, her geçerli hamle sonucu arttırılabileceği de belirlenebilmektedir. Oyun başladığı anda bir sayaç yardımıyla bu süreler zamana bağlı olarak azaltılmaktadır. Bu süreler countDown.js dosyası içerisindeki fonksiyonlar yardımıyla tutulmaktadır. Burada kullanılan ana fonksiyon, aşağıdaki **Tick** isimli fonksiyondur;

```
1  function Tick() {
2    if (GameController.GameOver == BOOL.TRUE){
3        TimerOn = BOOL.FALSE;
4        TimerOnOpp = BOOL.FALSE;
5        return;
6    }
7
8    if(timing != 0 & player == 1 & TotalSeconds <= 0 &
     TotalSecondsOpp <= 0){
9        TimerOn = BOOL.FALSE;
10       TimerOnOpp = BOOL.FALSE;
11       GameController.GameOver = BOOL.TRUE;
12       $("#GameStatus").text("OYUN BERABERE {Karşılıklı süre doldu}");
13       $('#NewGameButton').attr("disabled", false);
14       $('#Resign').attr("disabled", true);
15       return;
16   }else if (timing != 0 & TotalSeconds <= 0) {
17       TimerOn = BOOL.FALSE;
18       GameController.GameOver = BOOL.TRUE;
19       if(player == 2 ){
20           socket.emit('Time is up');
21       }else if(player == 1){
22           if(playerSide == COLOURS.WHITE){
23               $("#GameStatus").text("OYUN BİTTİ
                 {Süre doldu - Kazanan SİYAH}");
24           }else if (playerSide == COLOURS.BLACK){
25               $("#GameStatus").text("OYUN BİTTİ
                 {Süre doldu - Kazanan BEYAZ}");
26           }
27           $('#NewGameButton').attr("disabled", false);
28           $('#Resign').attr("disabled", true);
29       }
30       return;
31   }else if(timing != 0 & player == 1 & TotalSecondsOpp <= 0 ){
```

```
32          TimerOnOpp = BOOL.FALSE;
33          GameController.GameOver = BOOL.TRUE;
34          if(opponentSide == COLOURS.WHITE){
35              $("#GameStatus").text("OYUN BİTTİ
                {Süre doldu - Kazanan BEYAZ}");
36          }else if (opponentSide == COLOURS.BLACK){
37              $("#GameStatus").text("OYUN BİTTİ
                {Süre doldu - Kazanan SİYAH}");
38          }
39          $('#NewGameButton').attr("disabled", false);
40          $('#Resign').attr("disabled", true);
41          return;
42      }
43
44      if(TimerOn == BOOL.TRUE){
45          TotalSeconds -= 1;
46          CountDownTimer.innerHTML = UpdateTimer(TotalSeconds);
47      }
48      if(TimerOnOpp == BOOL.TRUE){
49          TotalSecondsOpp -= 1;
50          CountDownTimerOpp.innerHTML = UpdateTimer(TotalSecondsOpp);
51      }
52      setClearTime = window.setTimeout("Tick()", 1000);
53  }//End of Tick Function
```

Tick fonksiyonunun 2. satırında, oyunun bitip bitmediği kontrol edilmektedir. Şayet oyun bitmişse sayaçlar durdurulmakta ve fonksiyondan çıkılmaktadır (2. ve 6. satırlar arası). Şayet bir oyun süresi belirlenmişse, tek kişilik bir oyunsa ve her iki tarafında süresi tükenmişse (8. satır) 9. ve 15. satırlar arasındaki kod çalıştırılmaktadır. 9. ve 10. satırlarda, sayaçlar durdurulmaktadır. 11. satırda, oyunun bittiği belirtilmektedir. 12. satırda, karşılıklı süre dolduğu için oyunun berabere bittiğine dair bir mesaj yazdırılmaktadır. 13. ve 14. satırlarda, düğmeler uygun konuma alınmaktadır. 15. satırda da artık ihtiyaç kalmadığı için fonksiyondan çıkılmaktadır. Şayet bir oyun süresi belirlenmiş ve oyuncunun süresi tükenmişse (16. satır) 17. ve 30. satırlar arasındaki kod çalıştırılmaktadır. 17. satırda, oyuncunun sayacı durdurulmaktadır. 18. satırda, oyunun bittiği belirtilmektedir. Şayet oyun iki kişilik bir oyunsa (19. satır) sunucuya sürenin dolduğuna dair "Time is up" etiketli bir mesaj gönderilmektedir (20. satır). Şayet oyun tek oyunculu bir oyunsa (21. satır) oyunun bittiğini ve kimin kazandığını gösteren bir mesaj yazdırılmakta ve düğmeler uygun konuma alınmaktadır (22. ve 29. satırlar arası). 30. satırda, fonksiyondan çıkılmaktadır. Şayet oyunda bir süre

belirlenmişse, oyun tek kişilikse ve rakibin (bilgisayarın) süresi kalmamışsa (31. satır) 32. ve 42. satırlar arasındaki kod çalıştırılmaktadır. 32. satırda, bilgisayarın sayacı durdurulmaktadır. 33. satırda, oyunun bittiği belirtilmektedir. 34. ve 38. satırlar arasında, oyunun bittiği ve kazanan tarafın kim olduğu belirtilmektedir. 39. ve 40. satırlarda, düğmeler uygun konuma alınmaktadır. 41. satırda, fonksiyondan çıkılmaktadır. Bu fonksiyon için şu ana kadar ön kontroller yapılmıştır. Tick fonksiyonunun 44. ve 52. satırları arası fonksiyonun asıl işlevi ile ilgilidir. Şayet oyuncunun sayacı kapatılmamışsa (44. satır) oyuncunun toplam süresinden bir saniye eksiltilmekte (45. satır) ve oyun ekranında uygun yere yazdırılmaktadır (46. satır). Kalan oyun süresi ekrana yazdırılmadan önce **UpdateTimer** fonksiyonuyla belirli bir formata sokulmaktadır (46. satır). 48. ve 51. satırlar arasında aynı işlem rakip için de yapılmaktadır. Tek kişilik bir oyunsa rakibin kalan süresi (TotalSecondsOpp) bilgisayar tarafından tutulmaktadır. Oyun iki kişilikse rakibin kalan süresi hem bilgisayar tarafından tutulmakta hem de rakip tarafından gönderilmektedir. Rakip için tutulan bu sürelerde tutarsızlık olabileceği düşünülerek rakip tarafından gönderilen süre kullanılarak bilgisayar tarafından tutulan süre sürekli olarak güncellenmektedir. **Tick** fonksiyonu tarafından çağrılan **UpdateTimer** ve **UpdateTimer** fonksiyonu tarafından çağrılan **LeadingZero** fonksiyonu aşağıdaki gibi countDown.js dosyası içerisinde yer almaktadır;

```
 1  function UpdateTimer(sec) {
 2      var Seconds = sec;
 3      var Hours = Math.floor(Seconds / 3600);
 4      Seconds -= Hours * (3600);
 5      var Minutes = Math.floor(Seconds / 60);
 6      Seconds -= Minutes * (60);
 7      var TimeStr = LeadingZero(Hours) + ":" + LeadingZero(Minutes) + ":"
        + LeadingZero(Seconds)
 8      return TimeStr;
 9  }//End of UpdateTimer Function
10
11  function LeadingZero(Time) {
12      return (Time < 10) ? "0" + Time : + Time;
13  }
```

UpdateTimer fonksiyonuna kalan süre parametre olarak verilmektedir. Saniye cinsinden verilen bu parametre **LeadingZero** fonksiyonunun yardımıyla **UpdateTimer** fonksiyonu tarafından "saat : dakika : saniye" şeklinde gösterilecek

bir formata dönüştürülerek TimeStr değişkenine atanmaktadır ve fonksiyondan bu değişken döndürülmektedir.

Tick fonksiyonunun 20. satırında gönderilen "Time is up" etiketli mesajın sunucu tarafından karşı tarafa gönderilmesi gerekmektedir. Bunun için, aşağıda görüldüğü gibi, sunucu aldığı mesajı aynı şekilde karşı tarafa iletmektedir;

```
132   socket.on('Time is up', function(){
133         socket.opponent.emit('Time is up');
134   });
```

Rakip tarafından alınan "Time is up" mesajı karşı tarafın süresinin dolduğunu belirtmektedir ve buna karşılık aşağıdaki işlemler yapılmaktadır (index.js);

```
341 socket.on('Time is up', function(){
342
343    if(TotalSeconds > 0){
344      socket.emit('Have time', TotalSeconds);
345      TotalSecondsOpp = 0;
346      CountDownTimerOpp.innerHTML = UpdateTimer(TotalSeconds);
347      CountDownTimerOpp.innerHTML = UpdateTimer(TotalSecondsOpp);
348      GameController.GameOver = BOOL.TRUE;
349      if( playerSide == 0 ){
350          $("#GameStatus").text("OYUN BİTTİ {Beyaz yendi - Süre doldu}");
351      }else if( playerSide == 1 ){
352          $("#GameStatus").text("OYUN BİTTİ {Siyah yendi - Süre doldu}");
353      }
354    }else{
355      socket.emit('My time is up too');
356      TotalSeconds = 0;
357      TotalSecondsOpp = 0;
358      CountDownTimerOpp.innerHTML = UpdateTimer(TotalSeconds);
359      CountDownTimerOpp.innerHTML = UpdateTimer(TotalSecondsOpp);
360      GameController.GameOver = BOOL.TRUE;
361      $("#GameStatus").text("OYUN BERABERE {Karşılıklı süre doldu}");
362    }
363    $('#NewGameButton').attr("disabled", false);
364    $('#Resign').attr("disabled", true);
365 });
```

341. satırda "Time is up" etiketli mesaj alındığında anonim bir fonksiyon başlatılmaktadır. Şayet mesajı alan oyuncunun süresi hala varsa (343. satır) "Have time" etiketi ile kalan süresini sunucuya göndermektedir (344). Senkronizasyon problemi olabilme ihtimaline karşı taraf için tutulan sayaç sıfırlanmaktadır (345. satır). 346. ve 347. satırlarda, son sayaç durumu ekranda güncellenir ve 348. satırda oyunun bittiği belirtilmektedir. 349. ve 353. satırlar arasında oyunun galibi ve nasıl yendiğini belirten mesaj ekrana yazdırılmaktadır. Şayet mesajı alan oyuncunun kendisinin süresi de tükendiyse (354. satır) sunucuya "My time is up too" mesajı gönderilmektedir (355. satır). 356. ve 357. satırlarda her iki tarafın da sayaçları sıfırlanmaktadır (sayaçların eksiye düşme ihtimaline karşı). 358. ve 359. satırlarda sayaçların son durumu ekrana yazdırılmaktadır. 360. satırda oyunun bittiği belirtilmektedir. 361. satırda ise karşılıklı olarak süre dolduğundan dolayı oyunun berabere bittiği ekrana yazdırılmaktadır. 363. ve 364. satırlarda düğmeler uygun konuma alınmaktadır. Aşağıda görüldüğü üzere sunucu tarafından alınan "Have time" ve "My time is up too" etiketli mesajlar olduğu gibi karşı tarafa iletilmektedir (server.js). Karşı taraf tarafından alınan bu mesajlardan "Have time" etiketli mesaja aşağıdaki gibi işlem yapılmaktadır (index.js);

```
367    socket.on('Have time', function(data){
368      TotalSecondsOpp = data;
369      CountDownTimerOpp.innerHTML = UpdateTimer(TotalSeconds);
370      CountDownTimerOpp.innerHTML = UpdateTimer(TotalSecondsOpp);
371      GameController.GameOver = BOOL.TRUE;
372      if( playerSide == 0 ){
373         $("#GameStatus").text("OYUN BİTTİ {Siyah yendi - Süre doldu}");
374      }else if( playerSide == 1 ){
375         $("#GameStatus").text("OYUN BİTTİ {Beyaz yendi - Süre doldu}");
376      }
377      $('#NewGameButton').attr("disabled", false);
378      $('#Resign').attr("disabled", true);
379    });
```

367. satırda, "Have time" mesajı için bir anonim fonksiyon başlatılmaktadır. Bu fonksiyona mesajla birlikte gönderilmiş olan rakibin kalan süresi (data) parametre olarak verilmektedir. 368. satırda, "data" karşı tarafın süresini tutan değişkene atanmaktadır. 369. ve 370. satırda ekrandaki sayaç bilgileri güncellenmektedir. 371. satırda oyunun sona erdiği bildirilmektedir. 372. ve 376. satırlar arasında yenen tarafın rakip olduğu ve nasıl yendiği ekrana yazdırılmaktadır. 377. ve 378. satırlarda düğmeler uygun konuma alınmaktadır.

İstemci tarafından "My time is up too" mesajı alındığında aşağıdaki işlemler yapılmaktadır (index.js);

```
381    socket.on('My time is up too', function(){
382        TotalSeconds = 0;
383        TotalSecondsOpp = 0;
384        CountDownTimer.innerHTML = UpdateTimer(TotalSeconds);
385        CountDownTimerOpp.innerHTML = UpdateTimer(TotalSecondsOpp);
386        GameController.GameOver = BOOL.TRUE;
387        $("#GameStatus").text("OYUN BERABERE {Karşılıklı süre doldu}");
388        $('#NewGameButton').attr("disabled", false);
389        $('#Resign').attr("disabled", true);
390    });
```

"My time is up too" mesajı iki tarafında süresinin aynı anda tükendiği anlamına gelmektedir. Fonksiyonda sayaçlar sıfırlanmakta, oyun bitirilmekte, ekrana sayaçlar yazdırılmakta, oyunun bittiği ve neden bittiği ekrana yazdırılmakta ve düğmeler uygun konuma alınmaktadır.

4 GRAFİK KULLANICI ARA YÜZÜ (GUI - GRAPHICAL USER INTERFACE)

Grafik kullanıcı ara yüzü ile kullanıcıların eş zamanlı satranç oyununu oynayacağı görsel ortam kullanıcılara sunulmaktadır. Bu bölümde kullanıcıları karşılayan ara yüzün kodlaması ele alınmaktadır. Kullanıcılar eş zamanlı satranç oyunu uygulamasını servis eden web sunucuya bağlandığında sunucu kullanıcıları index.ejs dosyasına yönlendirmektedir. "ejs" (Embedded JavaScript) uzantılı bu dosya ile JavaScript kodları ve/veya bazı veriler kullanılarak, "ejs" dosyası yorumlanmakta ve bir html çıktı elde edilebilmektedir. Ancak eş zamanlı satranç oyununda bu özellik kullanılmamıştır. Burada ejs uzantılı dosyası node.js kütüphanesi ile uyum sağlanabilmesi için kullanılmaktadır. Oyun kurulumu esnasında ekran görüntüsü üzerinde yapılması gereken değişiklikler index.js dosyası yardımıyla index.ejs dosyasının sunduğu alanlara ait değerler değiştirilerek sağlanmaktadır.

4.1 Grafik Ara Yüzdeki Statik Alanlar

Grafik ara yüzde statik alanlara ait tanımlamalar index.ejs dosyası içerisinde yapılmaktadır. index.ejs dosyasının içeriği klasik bir html dosya içeriğidir.

index.ejs dosyası, html etiketleri içerisinde yer alan head ve body etiketleri içerisinde iki ana bölümden oluşmaktadır. head bölümü aşağıdaki gibidir;

```
1  <head>
2    <meta charset="utf-8">
3    <title>sChess</title>
4    <link href='http://fonts.googleapis.com/css?family=Tangerine'
      rel='stylesheet' type='text/css'>
5    <link rel="stylesheet" href="styles.css">
6    <script type="text/javascript"> if (!window.console) console =
      {log: function() {}}; </script>
7  </head>
```

head bölümünün 1. ve 7. satırlarında head bölümünün başladığı ve bittiği işaretlenmektedir. 2. satırında kullanılan karakter kümesi, 3. satırında başlık ve 4. satırında da kullanılacak olan bir font'un bulunduğu yere link verilmektedir. 5.satırında, web sayfasının görünümüyle ilgili düzenlemelerin yapıldığı styles.css dosyasına link verilmektedir. Ayrıca script tipi olarak javascript kullanılacağı 6. satırda belirtilmektedir.

index.ejs dosyasında yer alan body bölümü ise aşağıdaki şekildedir;

```
1  <body>
2    <div id="big_wrapper">
3      <header id="top_header">
4        <h1>Eş Zamanlı Satranç Oyunu</h1>
5      </header>
6
7      <nav id="top_menu">
8        <ul>
9          <li><a href="/">Ana sayfa</a></li>
10         <li><a href="rules.html">Kurallar</a></li>
11         <li><a href="about.html">Hakkımızda</a></li>
12         <li><a href="contacts.html">İletişim</a></li>
13       </ul>
14     </nav>
15
16     <div id='opponentWait'>
17       <h1>Yeni bir oyun kurdunuz.</h1>
18       <h1>Bir rakip oyuncunun oyuna katılması bekleniyor...</h1>
19     </div>
20
```

```
21      <div id="form">
22        <h2>Oyun kriterlerini belirleyiniz:</h2>
23
24          <div id="labels">
25            <ul>
26              <li><b>  İsminiz : </b> </li>
27              <li><b>Oyuncu Sayısı:</b></li>
28              <li><b class='mode'>Oyun Modu:</b></li>
29              <li><b
                class='playerSide'>Tarafınız:   </b></li>
30              <li><b class='timing'>Oyun Süresi: </b></li>
31              <li><b class='increment'>Artış Süresi: </b></li>
32            </ul>
33          </div>
34
35          <div id="inputs">
36            <form id = "selectionForm">
37            <ul>
38              <li><input id="userId" name="userId" maxlength="12"/>
                <input id="nickCheck" type="button" value="Kontrol Et"/></li>
39              <li>
40                  <input type="radio" id="player" name="player" value=1>
                    Tek Oyuncu
41                  <input type="radio" id="player" name="player" value=2 >
                    İki Oyuncu
42              </li>
43              <li>
44                  <input type="radio" class="mode" name="mode" value=1>
                    <span class='mode'>Öğrenme</span>
45                  <input type="radio" class="mode" name="mode" value=2>
                    <span class='mode'>Oyun</span>
46              </li>
47              <li>
48                  <input type="radio" class="playerSide" name="playerSide"
                    value=0><span class='playerSide'>Beyaz</span>
49                  <input type="radio" class="playerSide" name="playerSide"
                    value=1><span class='playerSide'>Siyah</span>
50              </li>
51              <li>
52                  <select name="timing" class="timing">
53                      <option value=0>Sınırsız</option>
54                      <option value=1>1 dakika</option>
55                      <option value=2>2 dakika</option>
56                      <option value=3>3 dakika</option>
57                      <option value=5>5 dakika</option>
```

```
58                      <option value=10>10 dakika</option>
59                      <option value=20>20 dakika</option>
60                      <option value=30>30 dakika</option>
61                      <option value=40>40 dakika</option>
62                      <option value=50>50 dakika</option>
63                      <option value=60>60 dakika</option>
64                  </select>
65              </li>
66              <li>
67                  <select name="increment" class="increment">
68                      <option value=0>Artışsız</option>
69                      <option value=1>1 saniye</option>
70                      <option value=2>2 saniye</option>
71                      <option value=3>3 saniye</option>
72                      <option value=4>4 saniye</option>
73                      <option value=5>5 saniye</option>
74                      <option value=6>6 saniye</option>
75                      <option value=7>7 saniye</option>
76                      <option value=8>8 saniye</option>
77                      <option value=9>9 saniye</option>
78                      <option value=10>10 saniye</option>
79                  </select>
80              </li>
81              <li>
82                    <input id="start" type="submit" value="Başlat"/>
83              </li>
84            </ul>
85            </form>
86          </div>
87
88          <p id='nickWarning'> </p>
89
90          <div class="waitingListWrapper">
91            <h3>Rakip Bekleyen Oyuncular:</h3>
92            <div class="waitingList"></div>
93            <div class="waitingListSide"></div>
94            <div class="waitingListTiming"></div>
95            <h4>Yukarıdaki rakip/rakiplerden birini seçerek oyunu
              başlatabilirsiniz veya "Başlat" düğmesine basarak yeni bir
              oyun kurabilirsiniz.</h4>
96          </div>
97      </div> <!--form div-->
98
99      <div id="wrapper">
```

```
100     <section id="main_section">
101       <div id="FenInDiv">
102         <input type="text" id="fenIn"/>
103         <button type="button" id="SetFen">Set FEN</button>
104       </div>
105       <div id = "side1"></div>
106       <div id="Board"></div>
107       <div id = "side0"></div>
108     </section>
109
110     <aside>
111       <div id="ThinkingTime">Bilgisayar Düşünme Süresi: <br/>
112         <select id="ThinkTimeChoice">
113           <option value="1">1s</option>
114           <option value="2">2s</option>
115           <option value="4">4s</option>
116           <option value="6">6s</option>
117           <option value="8">8s</option>
118           <option value="10">10s</option>
119         </select>
120       </div>
121
122       <div id="TimeInfo"><b>Oyun Süresi:</b> </div>
123
124       <div id="Timer"><span id="TimerUserName"><b>Sayaç </b>
          </span>
125         <span id="CountDownTimer"></span>
126       </div>
127
128       <div id="TimerOpp"><span id="TimerUserNameOpp"><b>
          Sayaç </b></span>
129         <span id="CountDownTimerOpp"></span>
130       </div>
131
132       <div id="MoveHistoryUser">
133         <label id="lbUserMoves" for="playerMoves"></label>
134         <textarea id="playerMoves" disabled></textarea>
135       </div>
136
137       <div id="MoveHistoryEngine">
138         <label id="lbEngineMoves" for="opponentMoves"></label>
139         <textarea id="opponentMoves" disabled></textarea>
140       </div>
141
```

```
142         <div id="Chat">
143           <label id="lbMessages" for="Messages">Mesajlar</label>
144           <div id="Messages" ></div>
145           <select name="MessageOptions" class="MessageOptions">
146             <option value=10 selected="selected">Lütfen oynar mısın artık?
                </option>
147             <option value=20>Hadi oyna kardeşim!</option>
148             <option value=30>Yeni oyuna var mısın?</option>
149             <option value=40>Tamam. Oynuyorum.</option>
150             <option value=50>Başka bir oyun kuruyorum.</option>
151             <option value=60>Beraberliği kabul ediyor musun?</option>
152             <option value=70>Evet</option>
153             <option value=80>Hayır</option>
154             <option value=90>Oyundan çıkıyorum.</option>
155             <option value=100>İşim çıktı. Sonra Görüşürüz.</option>
156             <option value=110>Tamam. Görüşürüz.</option>
157             <option value=120>Artık başka zaman yenilirsin :)</option>
158             <option value=130>Tamam.</option>
159             <option value=140>Kabul ediyorum.</option>
160             <option value=150>Kabul etmiyorum.</option>
161           </select>
162           <button type="button" id="SendMessage">Gönder</button><br/>
163         </div>
164
165         <div id="Buttons">
166           <button type="button" id="NewGameButton">Tekrar Oyna
              </button><br/>
167           <button type="button" id="MoveButton">Oynat</button><br/>
168           <button type="button" id="ForwardButton">İleri al</button><br/>
169           <button type="button" id="TakeButton">Geri al</button><br/>
170           <button type="button" id="Resign">Oyundan Çekil</button><br/>
171           <span id="GameStatus"></span>
172         </div>
173       </aside>
174     </div> <!--wrapper div-->
175
176     <footer id="the_footer">
177       Copyright© UBE 2015
178     </footer>
179
180   </div><!--big_wrapper div-->
181
182   <script src="js/jquery-1.11.3.min.js"></script>
183   <script src="/socket.io/socket.io.js"></script>
```

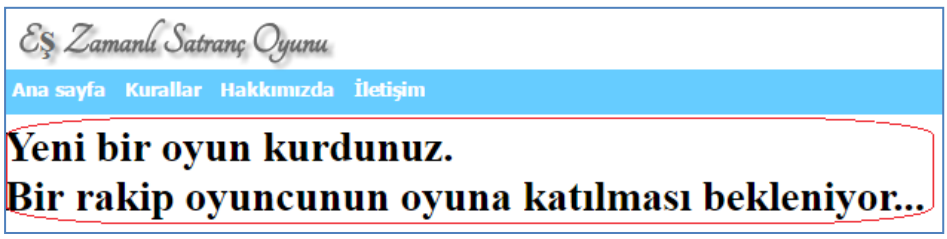
```
184     <script src="js/defs.js"></script>
185     <script src="js/countDown.js"></script>
186     <script src="/js/index.js"></script>
187     <script src="js/io.js"></script>
188     <script src="js/board.js"></script>
189     <script src="js/movegen.js"></script>
190     <script src="js/makemove.js"></script>
191     <script src="js/evaluate.js"></script>
192     <script src="js/pvtable.js"></script>
193     <script src="js/search.js"></script>
194     <script src="js/perft.js"></script>
195     <script src="js/gui.js"></script>
196     <script src="js/main.js"></script>
197 </body>
```

Body bölümü kullanıcının ekranda gördüğü görüntünün şablonunun olduğu bölümdür. Bu şablona ait değerler oyun kurulumu ve oyun esnasında js uzantılı JavaScript dosyaları vasıtasıyla aktif olarak değiştirilmektedir. Body bölümünün 1. satırında, body etiketi kullanılarak bu bölümün body bölümü olduğu belirtilmektedir. 2. satırda, big_wrapper isimli bir div oluşturulmaktadır. div'ler bir bölümdeki kodları gruplama amacıyla kullanılmaktadır. div'lere ve diğer (header, nav, aside vb.) gruplama amacıyla kullanılan konteynır tiplerine id'ler (tanımlayıcı) ve/veya class'lar (sınıflandırıcı) tanımlanmaktadır. Bu id ve class'lar index.js dosyası içerisinden çağrılmakta ve interaktif bir kullanıcı ara yüzü oluşturulmaktadır. 3. ve 5. satırlar arasında, oyunun başlık kısmı yer almaktadır. Bu alanın ekran görüntüsü Şekil 4-1'nin "1" rakamı ile işaretlenmiş olan kısmında olduğu gibi görülmektedir.



Şekil 4-1 Kullanıcı ara yüzü - karşılama ekranı

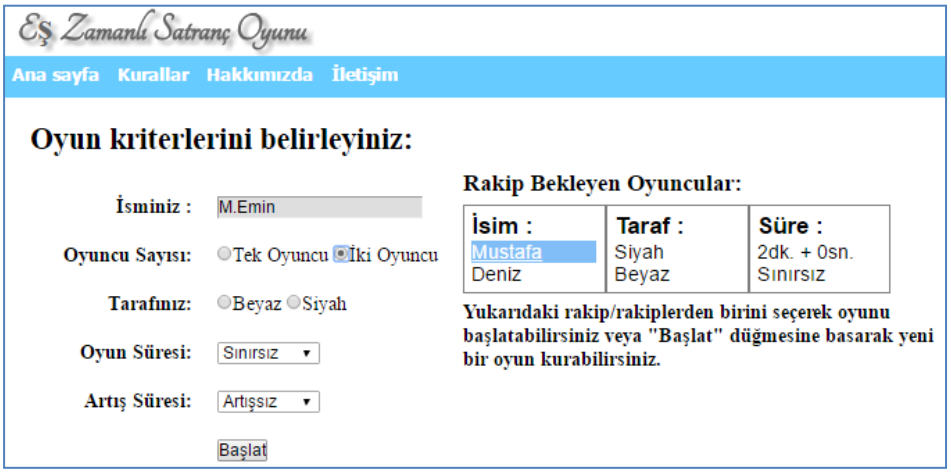
7. ve 14. satırlar arasında, kullanıcının web sayfası üzerinde gezinmek için kullanılan ve top_menu id'si ile tanımlanan gezinme çubuğuna (navigation bar) ait kodlar yer almaktadır. Gezinme çubuğunun görüntüsü Şekil 4-1'nin "2" ile işaretlenmiş kısmında olduğu gibidir. Bu alana ve web sayfası üzerinde bulunan diğer tüm alanlara ait şekil düzenlemesi (konumu, boyutu, rengi, görünürlüğü vb.) styles.css dosyası ile yapılmaktadır. 16. ve 19. satırlar arasında, bir oyuncunun ağ üzerinden başka bir oyuncuyla oynamak için bir oyun kurduğunda rakibi beklerken karşısına çıkacak olan mesaj yer almaktadır. Şekil 4-2'de bu mesajın ekranda nasıl çıkacağı görülmektedir. Bu alanın id'si opponentWait olarak belirlenmiştir.



Şekil 4-2 Kullanıcı ara yüzü - rakibin beklenmesi

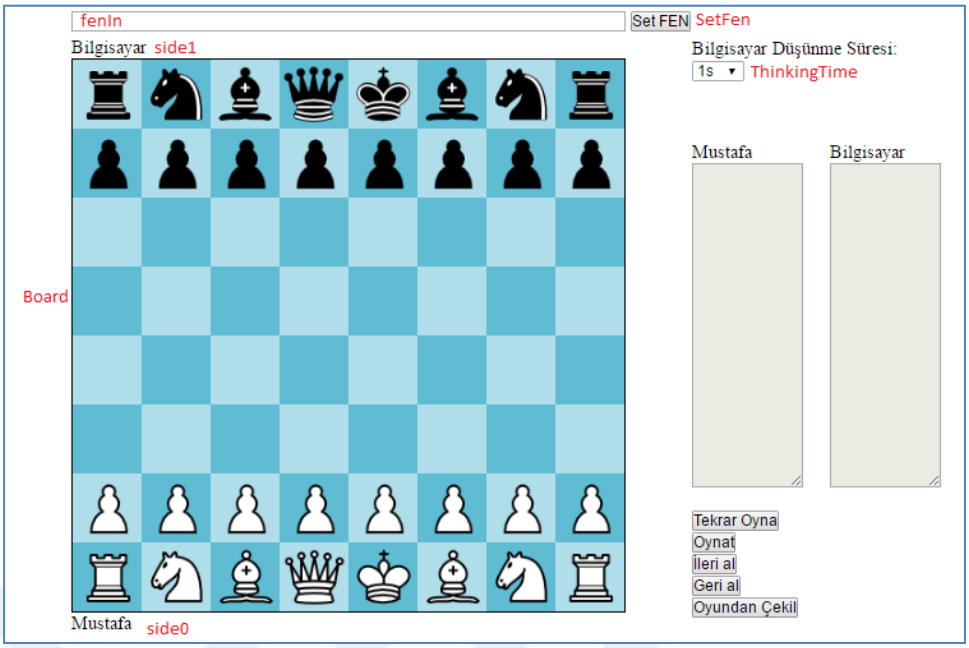
21. ve 97. satırlar arasında, yeni bir oyun için kullanıcının girdiği bilgiler bir form aracılığıyla alınmaktadır. Bu formun ekran görüntüsü Şekil 4-1'de "3" rakamı ile işaretlenmiş olan alanda olduğu gibidir. 24. ve 33. satırlar arasında yer alan labels id'li div alanında kullanıcıdan alınan bilgilerin etiketleri yer almaktadır. labels div alanında etiketlerin karşılığında kullanıcının girdi yapacağı

alanlarsa 35. ve 86. satırlar arasındaki inputs id'li div alanında yer almaktadır. inputs div'inde 36. satırda bir form başlatılmaktadır. Bu form sayesinde kullanıcıdan alınan bilgiler sunucuya gönderilmektedir. inputs div'i içerisindeki 38. satırdaki alan, labels div'inin ilk alanı olan "İsminiz" alanına karşılık gelmektedir. Kullanıcı ismini bu alana girmektedir. Yine 38. satırda bu ismin daha önce alınmış olup olmadığını kontrol etmek için bir "Kontrol et" düğmesi bulunmaktadır. 39. ve 42. satırlar arasında, labels div'indeki "oyuncu sayısı"na karşılık gelen alan (id = player) bulunmaktadır. Bilgisayara karşı mı (Tek Oyuncu) yoksa ağ üzerinde başka bir oyuncuya karşı mı oynanacağı (İki Oyuncu) bu alanda seçilmektedir. 43. ve 46. satırlar arasında, oyun modunun ne olacağı seçilmektedir. "Öğrenme" ve "Oyun" olmak üzere iki farklı oyun modu bulunmaktadır. Öğrenme modu sadece "Tek Oyuncu"lu oyunlarda geçerlidir ve oyun esnasında hamleleri geri/ileri alma özellikleri mevcuttur. 47. ve 50. satırlar arasında oyuncu hangi taraf (siyah/beyaz) olacağını belirlemektedir. Beyaz value = 0 ile siyah ise value = 1 ile temsil edilmektedir. 51. ve 65. satırlar arasında, oyun süresinin belirlenebileceği bir listeye ait kodlar yer almaktadır. 66. ve 80. satırlar arasında da her doğru hamle sonucunda oyun süresine eklenecek süreye ait kod yer almaktadır. Bu iki alan kullanılmak istenmiyorsa value = 0 ile temsil edilen "Sınırsız" ve "Artışsız" seçenekleri seçilebilmektedir. Herhangi bir süre (timing) belirlenmezse doğal olarak artış miktarı (increment) da belirlenememektedir. Tek oyuncu ve öğrenme modu seçeneklerinde de oyun süresi devre dışı bırakılmaktadır. selectionForm içerisinde son olarak 81. ve 83. satırlar arasında yer alan "Başlat" düğmesi bulunmaktadır. Bu düğmeye basıldığında yapılacak olan işlemler index.js dosyası içerisinde yer almaktadır. 88. satırda, nickWarning id'sine sahip bir paragraf alanı yer almaktadır. Şayet kullanıcı tarafından belirlenen isim daha önce alınmışsa Bu ismin alındığını gösteren uyarı mesajı bu alana yazdırılmaktadır. 90. ve 96. satırlar arasında ise bir oyun kurmuş olan ancak henüz ağ üzerinde bir rakiple eşleşmemiş olan oyuncuların ismi, tarafı, kurduğu oyunun süresi ve artış miktarını gösteren bir listeye ait bir alan bulunmaktadır. Bu alan Şekil 4-3'de "Rakip bekleyen oyuncular" listesindeki ekranda görülmektedir.



Şekil 4-3 Rakip bekleyen oyuncular

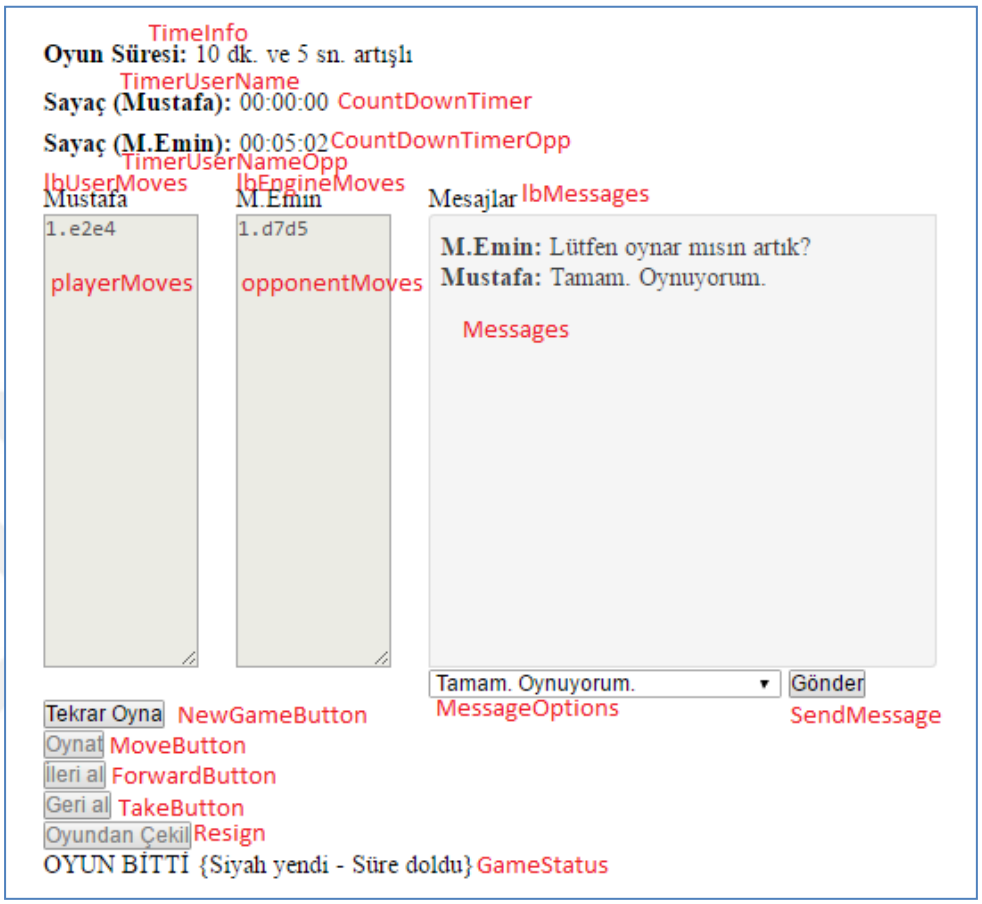
Kullanıcıdan alınacak olan veriler ile ilgili kısma ait statik alan tanımlamaları şu ana kadar tamamlanmıştır. Tek oyunculu bir oyun belirlenmişse "Başlat" düğmesine basılır basılmaz oyun başlamaktadır. İki oyunculu oyunlar ise rakibin oyun kuran oyuncuyu listeden seçmesinden sonra başlamaktadır. Bu aşamadan sonra kullanıcının veri girdiği alan ekrandan kaldırılmakta ve oyunun oynanacağı oyun tahtasının ve diğer bileşenlerin bulunduğu bölüm ekrana getirilmektedir. Bu bölümün statik kodlaması 99. ve 174. satırlar arasında yer alan wrapper id'li div içerisinde yer almaktadır. Burada 100. ve 108. satırlar arasında oyun tahtasının yer alacağı section etiketli main_section id'sine sahip kısım bulunmaktadır. 101. ve 104. satırlar arasında FEN metnine ait metin giriş alanı ve "Set FEN" düğmesi bulunmaktadır (Bkz.4.3). 105. satırdaki "side1" id'li div alanına rakibin ismi yazılmaktadır. 106. Satırda bulunan "Board" id'li div alanına oyun tahtası yerleştirilmektedir. 107. satırda bulunan "side0" id'li div alanına ise oyuncunun ismi yazılmaktadır. 110. ve 173. satırlar arasında oyuna ait yan bilgilerin yer aldığı aside kısmı yer almaktadır. 111. ve 120. satırlar arasında sadece Tek oyunculu ve "Öğrenme" modunda aktif edilen bilgisayarın düşünme süresi yer almaktadır. Bu süre bilgisayarın en iyi hamleyi bulması için bilgisayara verilen süredir. Oyun esnasında kullanıcı tarafından değiştirilebilmektedir. Bu alan diğer oyun modlarında bulunmamaktadır. Çünkü, iki oyunculu oyunlarda zaten bilgisayarla oynanılmamaktadır. Tek oyunculu oyun modunda ise o alanda, kalan oyun süresini gösteren sayaç yer almaktadır. Yukarıda anlatılan alanların ekranda nerelerle eşleştiği Şekil 4-4'de görülebilmektedir.



Şekil 4-4 Oyun tahtası üzerindeki alanların görünümü

122. satırda "Oyun Süresi" yazısı TimeInfo id'li div içerisinde yer almaktadır. Bu alana seçilen oyun süresi oyun başlamadan hemen önce eklenmektedir. 124. ve 126. satırlar arasında oyuncuya ait sayaç bilgisinin yer alacağı "Timer" id'li bir div alanı bulunmaktadır. Bu alan içerisindeki "TimerUserName" id'li alana, oyun başlamadan hemen önce "Sayaç" yazısının yanına parantez içerisinde oyuncunun ismi yazılmaktadır. "CountDownTimer" id'li alana ise oyun esnasındaki mevcut sayaç bilgisi yazdırılmakta ve sürekli güncellenmektedir. Benzer şekilde rakibin sayacına ait işlemler de 128. ve 130. satırlar arasında yapılmaktadır. 132. ve 135. satırlar arasında bulunan bölümde oyuncu tarafından oyun süresince yapılmış hamleler olan hamleler yer almaktadır. Oyuncu hamle yaptıkça "playerMoves" id'li metin alanı güncellenmektedir. 137. ve 140. satırlar arasındaki bölümde de aynı işlem rakip için yapılmaktadır. 142. ve 163. satırlar arasında, oyuncuların kısıtlı da olsa haberleşebilmesi için bir haberleşme alanı yer almaktadır. Herhangi bir yasal sorun ile karşılaşılmaması için bu haberleşme alanında, daha önceden belirlenmiş kısıtlı bir metin kümesiyle haberleşme yapılabilmektedir. 165. ve 172. satırlar arasında "Tekrar Oyna", "Oynat", "İleri al", "Geri al" ve "Oyundan Çekil" düğmeleri yer almaktadır. Düğmelerin altında da oyunun sonucunu gösteren GameStatus id'li span etiketli bir alan bulunmaktadır (171. satır). Oynat düğmesine basıldığında bilgisayar her iki taraf için de karar verip hamle yapmaktadır. Bu düğmelerin hepsi öğrenme modunda aktif konumdadır. Diğer modlardaki durum oyunun şekkine göre belirlenmektedir. 176. ve 178. satırlar arasındaki kısım şu an için

kullanılmamaktadır. İleride kullanılması düşünüldüğünde aktif edilecektir. Yukarıda anlatılan aside bölümüne ait alanlar ve tanımlayıcıları Şekil 4-5'de görülmektedir.



Şekil 4-5 Oyun tahtasındaki yardımcı alanlar (aside)

182. satır ve 197. satırlar arasında, eş zamanlı satranç oyununun çalıştırılması için gerekli olan javascript kodları index.ejs dosyasına dahil edilmektedir. 182. satırda jQuery kütüphanesinin kullanılabilmesi için jquery-1.11.3.min.js dosyası index.ejs dosyasına dahil edilmektedir. 183. satırda, sunucu ve istemciler arasında iletişimin sağlanabilmesi için kullanılan socket.io kütüphanesi index.ejs dosyasına dahil edilmektedir. 184. ve 196. satırlar arasında ise eş zamanlı satranç oyununda kullanılan kodlar yer almaktadır.

4.2 Grafik Ara Yüzün Dinamik Alanları ve Yeni Oyun Kurulumu

Grafik ara yüzün oyun kurulum esnasında kullanıcı etkileşimi ve oyunun uygun şekilde başlatılması için index.js dosyası, sunucu tarafındaki işlemler için

de server.js dosyası kullanılmaktadır. index.js dosyası içerisinde bir **jQuery** fonksiyonu, aşağıdaki gibi 1. satırda başlatılmakta ve 401. satırda da yine aşağıdaki gibi sonlandırılmaktadır. Bu kod bir **jQuery** kodudur ve html dokümanının yüklenmesi tamamlandığında bu anonim fonksiyonu başlat anlamına gelmektedir. Bu tür fonksiyonlara olay tetiklemeli (event-trigerred) fonksiyonlar denilmektedir ve eş zamanlı satranç oyunu kodlamasında sıklıkla kullanılmaktadır.

```
  1 $(document).ready(function(){
401 });//document.ready()
```

Kodun tamamının aynı anda incelenmesi zor olacağından kısım kısım incelenecektir. İlk incelenecek olan kısım aşağıdaki ilk 13 satırdır;

```
 1 $(document).ready(function(){
 2
 3     var wListLength = 0;
 4
 5
 6     disableButtons();
 7     $('#nickCheck').click( function(){
 8         checkNick();
 9     });
10
11     $('#userId').change(function(){
12         checkNick();
13     });
```

3. satırda, rakip oyuncu bekleyen oyuncuların listesini tutmak için kullanılan bir değişken tanımı yapılmaktadır. 6. satırda aşağıdaki **disableButtons** fonksiyonu çağrılarak başlangıçta aside kısmında yer alan tüm düğmeler kullanıma kapatılmaktadır. Daha sonra program içerisinde ihtiyaç duyulan düğmeler kullanıma açılmaktadır. **disableButtons** fonksiyonu index.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
138 function disableButtons(){
139     $('#NewGameButton').attr("disabled", true);
140     $('#MoveButton').attr("disabled", true);
141     $('#ForwardButton').attr("disabled", true);
142     $('#TakeButton').attr("disabled", true);
```

```
143     }
```

index.js dosyasının 8. ve 12. satırlarında **checkNick** fonksiyonu çağrılarak belirlenen oyuncu isminin geçerli olup olmadığı kontrol edilmektedir. 8. satırdaki çağrımı KontrolEt (id = nickCheck) düğmesine basıldığında (7. satır), 12. satırdaki fonksiyon çağrımı ise userId (ismin girildiği text alanı)'de bir değişiklik olduğunda (11.satır) yapılmaktadır. userId alanındaki değişiklik, o alana bir veri girilmesi ve alandan çıkıldıktan sonra değişiklik olarak algılanmaktadır. **checkNick** fonksiyonu index.js içerisinde aşağıdaki şekilde yer almaktadır;

```
152   function checkNick(){
153     userName = cleanInput($('#userId').val().trim());
154     if( userName != ''){
155           socket.emit('new player', userName, function(data){
156             if(data){
157                   $('#nickWarning').html('Geçerli Kullanıcı İsmi!');
158                   $('#userId').attr("readonly", true);
159                   $('#userId').css('background-color' , '#DEDEDE');
160                   $('#nickCheck').hide();
161                   if(player == 2 & wListLength > 0){
162                         $('#nickWarning').html('');
163                         $('.waitingListWrapper').show();
164                   }
165
166             }else if(!data){
167                   $('#nickWarning').html('Bu isim daha önce alınmış.
                      Lütfen başka isim deneyiniz!');
168                   userName = '';
169             }
170           });
171     }else{
172           $('#userId').val('');
173           $('#nickWarning').html('Lütfen bir isim giriniz!');
174     }
175   }//end of checkNick
```

153. satırda, userId kısmındaki değerin varsa başındaki ve sonundaki boşluklar silinerek **cleanInput** fonksiyonuna girdi olarak verilmektedir. **cleanInput** fonksiyonunun döndürdüğü değer de userName değişkenine atanmaktadır. **cleanInput** fonksiyonunun kullanılma amacı, herhangi bir injection saldırısını önlemek içindir ve kod aşağıdaki gibidir;

```
397     function cleanInput (input) {
398         return $('<div/>').text(input).text();
399     }
```

154. satırda, userName değişkeninde bir değer olup olmadığı kontrol edilmektedir. Bir değer varsa 155. satıra geçilmektedir. 155. satırda, **socket.io** kütüphanesinin bir fonksiyonu olan **socket.emit** fonksiyonu kullanılmaktadır. **socket.emit** fonksiyonu "new player" etiketini kullanarak oyuncu tarafından belirlenmiş olan ismi (userName) sunucuya göndermekte ve sunucudan gelecek olan cevabı (data) işlemek için anonim bir fonksiyon başlatmaktadır. userName'in sunucuya gönderilme sebebi; bu ismin başka bir oyuncu tarafından alınıp alınmadığının kontrolünü yapmaktır. Bu kontrol sunucu tarafında aşağıdaki şekilde yapılmaktadır (server.js dosyası içerisinde);

```
36  io.sockets.on('connection', function(socket){
37      socket.on('new player', function(data, callback){
38          if(data in allPlayers){
39              callback(false);
40          }else{
41              socket.nickname = data;
42              allPlayers[socket.nickname] = socket;
43              io.sockets.emit('waitingList', waitingList);
44              callback(true);
45          }
46      });
```

Sunucuya bir bağlantı yapıldığında 36. satırdaki anonim fonksiyona o istemcinin bağlantı yaptığı soket verisi parametre olarak verilerek anonim fonksiyon başlatılmaktadır. O soketten "new player" etiketli bir veri geldiğinde data ve callback isimli parametreler alan bir anonim fonksiyon daha başlatılmaktadır. data istemci tarafından belirlenmiş olan ve gönderilen kullanıcı ismi bilgisini içermektedir. callback ise istemciye dönülecek olan veriyi temsil etmektedir. 38. satırda, allPlayers isimli aktif tüm kullanıcıların isminin tutulduğu dizi taranmaktadır. allPlayers dizisinin elemanlarından birisi data parametresinin değeriyle aynı değeri taşıyorsa 39. satırda istemciye false verisi gönderilmektedir. Şayet bu isim ilk defa alınıyorsa bir eşleşme olmayacak ve 41. satırdan devam edilecektir. 41. satırda o soket numarasına nickname isimli bir alt değişken bağlanmakta (nesne özelliği) ve yeni isim o değişkene atanmaktadır. Böylelikle her istemcinin ismi bağlı olduğu soket numarası ile işleştirilerek tutulmaktadır. 42. satırda, oyuncunun ismi allPlayer dizisinde bir elemanın indis numarasını (ismini)

oluşturmakta ve bağlandığı soket numarası da o alana kaydedilmektedir. Böylelikle bir ismin daha önce alınıp alınmadığı sadece indis isimleri kontrol edilerek ("data in allPlayer" şeklinde) bulunabilmektedir. 43. satırda, rakip bekleyen oyuncuların listesi (waitingList) tüm istemcilere gönderilmektedir. Bunun amacı, hem yeni bağlanan istemcinin bu veriyi edinmesini sağlamak hem de waitingList'te bir güncelleme varsa diğer istemcileri de bundan haberdar etmektir. 44. satırda, belirlenen ismin geçerli bir isim olduğu kullanıcıya "true" mesajı gönderilerek bildirilmektedir.

index.js dosyası incelenmeye devam ediliyor; Sunucudan bir cevap geldiğinde bu cevabın (data) hangi değeri içerdiği (true/false) 156. satırda kontrol edilmektedir. Sunucu tarafından true değeri gönderilmişse yani, belirlenen isim başka bir oyuncu tarafından kullanılmıyorsa 157. satırdan devam edilmektedir. 157. satırda, nickWarning id'li alana uygun bildirim metni yazdırılmaktadır. Belirlenen ismin tekrar değiştirilmemesi için userId alanı sadece okunabilir duruma getirilmektedir ve zemin rengi olarak da gri bir ton yapılmaktadır (158. ve 159. satırlar). Artık bir isim belirlendiği için "Kontrol Et" düğmesine ihtiyaç kalmadığından dolayı o düğme ekrandan kaldırılmaktadır (160. satır). 161. satırda, oyunun iki oyunculu bir oyun olup olmadığı ve rakip bekleyen herhangi bir oyuncu olup olmadığı sorgulanmaktadır. Şartlar uyuyorsa 162. satırda nickWarning kısmının içeriği silinmektedir. Çünkü o alana rakip bekleyen oyuncuların listesi yazdırılmaktadır. 163. satırda waitingListWrapper kısmı ekranda gösterilecek duruma getirilmektedir. Böylelikle yeni oyuncu, iki kişilik oyun oynamak isteyen oyuncuların listesini görebilmektedir. Şayet sunucu tarafından gönderilen cevap false ise 157. satırdan değil de 166. satırdan devam edilmektedir. 167. satırda belirlenen ismin daha önceden alındığına dair bir mesaj yazdırılmaktadır. 168. satırda da yeni ismin girilmesi için userName değişkeninin içeriği silinmektedir. userName alanı boş bırakılmışsa 154. satırdan 171. satıra geçilmektedir. 172. satırda userId alanının içeriği silinmektedir. Çünkü bu alanda boşluk karakteri girilme ihtimali mevcuttur. 173. satırda da bir isim girilmesi için bir uyarı yazdırılmaktadır.

Grafik kullanıcı ara yüzünde bir isim belirlendikten sonra oyuncu sayısının belirlenmesi gerekmektedir. Bu kısım ile ilgili kodlar 83. satır ile 109. satırlar arasında yer almaktadır.

```
83    $('input[name="player"]').change(function(e) {
84        player = $( this).val();
```

```
85          if (userName != ''){
86              if(player == 2 ){
87                  $('.timing').show();
88                  $('.increment').show();
89                  $('.mode').hide();
90                  if( wListLength > 0 ){
91                      $('#nickWarning').html('');
92                      $('.waitingListWrapper').show();
93                  }
94              }else if(player == 1){
95                  $('.timing').hide();
96                  $('.increment').hide();
97                  $('.mode').show();
98                  if (mode == 2){
99                      $('.timing').show();
100                     $('.increment').show();
101                 }
102                 $('.waitingListWrapper').hide();
103             }
104         }else{
105             $('.waitingListWrapper').hide();
106             $('#userId').val('');
107             $('#nickWarning').html('Lütfen bir isim
                giriniz!');
108         }
109     });//player change
```

83. satırda, player (Oyuncu sayısı) ile tanımlanan radio button alanında bir değişiklik olduğunda bu satırdaki anonim fonksiyon çağrılmaktadır. 84. satırda değişikliğin yapıldığı alandaki değer player değişkenine atanmaktadır. Yani 1 veya 2 değeri player değişkenine atanmaktadır. Oyuncu sayısının belirlenmesinden önce bir oyuncu isminin belirlenmiş olması gerektiğinden dolayı 85. satırda bu kontrol yapılmaktadır. Şayet bir oyuncu ismi belirlenmişse 86. satıra geçilmektedir. İki oyunculu bir oyun seçimi yapılmışsa 87. satırdan devam edilmektedir. 87. ve 88. satırlarda oyun süresi ve artış miktarı seçimlerinin yapılabilmesi için ilgili alanlar görünür hale getirilmektedir. Bu alanlar başlangıç durumunda görünmez haldedir ve başlangıç ayarları style.css dosyası içerisinde yer almaktadır. 89. satırda oyun modu seçimi ile ilgili alan görünmez yapılmaktadır. Çünkü ikili oyunlarda öğrenme modu bulunmamaktadır. Sadece oyun modu bulunmaktadır. 90. satırda, daha önceden bir ikili oyun kurulumu yapmış ve rakip bekleyen oyuncu olup olmadığı kontrol edilmektedir. Bu kontrol

sunucu tarafından gönderilen wListLength değerine bakılarak yapılmaktadır. Şayet rakip bekleyen oyuncu varsa 91. satırda, o alandaki uyarı kaldırılmakta ve 92. satırda, rakip bekleyen oyuncu listesi görünür hale getirilmektedir. Şayet geçerli bir oyuncu ismi belirlenmiş ve tek oyunculu bir oyun kuruluyorsa 94. satırdan devam edilmektedir. 95. ve 96. satırlarda, oyunun modunun "öğrenme modu" olduğu varsayılarak zaman belirleme ile ilgili alanlar görünmez yapılmaktadır. Bu ve benzer kodlamalar, oyuncunun tercihlerini belirlerken seçenekler arasında gezinmesi sonucu bir önceki seçimine ait bilgilerin ekrandan silinmesi için kullanılmaktadır. 97. satırda, oyun modu ile ilgili seçip yapılan kısım görünür hale getirilmektedir. Oyun modunun belirlendiği alan olay tetiklemeli bir alan olduğu için buradaki kod akışından bağımsız olarak oyun modu belirlenebilmektedir. Bu durumda 98. satırdaki kontrol yapılarak oyun modu 2 yapılmışsa oyun süreleriyle ilgili kısım görünür hale getirilmektedir (99. ve 100. satırlar). Tek oyunculu bir oyun kurulumu yapıldığından dolayı 102. satırda rakip bekleyen oyuncuların listesi görünmez yapılmaktadır. 85. satırdaki if sorgusu false değer döndürürse 104. satırdan devam edilmektedir. 105. satırda rakip bekleyen oyuncu listesi gizlenmektedir. Çünkü oyuncunun bir isim belirlemeden rakip bekleyen oyuncuları görmesinin uygun olmayacağı değerlendirilmektedir. 106. satırda, userId alanı boşaltılmaktadır. Bu alan zaten boş olduğu için zaten bu satıra geçildi diye düşünülebilir ancak bu alanda boşluk karakteri varsa onların temizlenmesi için eklenmiş bir satırdır. 107. satırda da bir isim belirlenmesi için kullanıcıya mesaj verilmektedir.

Oyuncu sayısının belirlenmesinin ardından tek oyunculu oyunlarda oyun modunun seçilmesi gerekmektedir (116. ve 127. satırlar arasındaki işlemler).

```
116     $('input[name="mode"]').change(function(e) {
117             mode = $( this).val();
118             if( mode == 1 ){
119                     $('.timing').hide();
120                     $('.increment').hide();
121                     enableButtons();
122             }else if ( mode == 2 ){
123                     $('.timing').show();
124                     $('.increment').show();
125                     disableButtons();
126             }
127     });
```

"mode" tanımlayıcısıyla tanımlanmış olan alanda bir değişiklik olduğu zaman bir anonim fonksiyon başlatılmaktadır (116. satır). "mode" alanındaki (radio button) değer mode değişkenine atanmaktadır (117. satır). Şayet mod değeri "1" seçilmişse 119. satırdan devam edilmektedir. Mod 1 öğrenme modu olduğu için ve öğrenme modunda oyun süresi tutulmadığı için 119. ve 120. satırlarda oyun süreleri ile ilgili kısım gizlenmektedir. Ayrıca 121. satırda **enableButtons** fonksiyonu çağrılarak tüm düğmeler aktif hale getirilmektedir. Böylece oyuncu istediği gibi oyunu ileri geri alabilmektedir. Şayet mod 2 seçilmişse 123. ve 124 satırlarda oyun süreci ile ilgili alanlar görünür hale getirilmekte ve düğmeler pasif hale getirilmektedir. Böylelikle oyuncunun gerçek oyun oynarken keyfi olarak oyunu ileri geri alması engellenmiş olmaktadır. **enableButtons** fonksiyonu aşağıdaki gibi kodlanmıştır;

```
129     function enableButtons(){
130         $('#NewGameButton').attr("disabled", false);
131         $('#MoveButton').attr("disabled", false);
132         $('#ForwardButton').attr("disabled", false);
133         $('#TakeButton').attr("disabled", false);
134     }
```

Tarafların belirlenmesi aşağıdaki şekilde olmaktadır;

```
111     $('input[name="playerSide"]').change(function(e) {
112             playerSide = $( this).val();
113             opponentSide = playerSide^1;
114     });
```

111. satırda, playerSide alanında herhangi bir değişiklik olduğunda bir anonim fonksiyon başlatılmaktadır. Burada belirlenmiş olan değer playerSide değişkenine atanmaktadır. Bu değer artık oyuncunun hangi tarafta oynadığını belirlemektedir (112. satır). 113. satırda, karşı tarafın rengi, opponentSide değişkenine playerSide değişkenindeki değerin tersi alınarak atanmaktadır (bir binary değerin 1 ile XOR'lanması o değerin tersi sonucunu vermektedir).

Oyun kurulumu için son olarak da oyun süresinin ve artış miktarının belirlenmesi gerekmektedir. Bu kısımlar aşağıdaki şekilde kodlanmıştır;

```
143     $('select[name="timing"]').change(function(e) {
144             timing = $( this).val();
```

```
145     });
146
147     $('select[name="increment"]').change(function(e)
        {
148             increment = $( this).val();
149     });
```

143. satırda, oyun süresi ile ilgili alanda bir değişiklik olduğunda bir anonim fonksiyon başlatılmaktadır. 144. satırda, belirlenmiş olan süre timing değişkenine atanmaktadır. 147. satırda, her geçerli hamle başına oyun süresine eklenecek olan süre ile ilgili kısımda bir değişiklik olduğunda anonim bir fonksiyon başlatılmaktadır. 148. satırda, belirlenen artırım süresi incerement değişkenine atanmaktadır. Şu ana kadar yeni bir oyun kurulumu ile ilgili alanlar incelenmiş bulunmaktadır. Bu aşamada artık hazırlanan form "Başlat" düğmesine basılarak selectionForm submit edilmektedir. Bu kısım index.js dosyasında 15. ve 81. satırlar arasında aşağıda şekilde yer almaktadır;

```
15      $('#selectionForm').submit(function(e){
16
17          e.preventDefault();
18          if( userName != '' & player == 1 ){
19              if (playerSide == COLOURS.BLACK){
20                  GameController.BoardFlipped = BOOL.TRUE;
21                  SetInitialBoardPieces();
22              }
23              $('#side1').append('Bilgisayar');
24              $('#side0').append(userName);
25              $('#lbUserMoves').append(userName);
26              $('#lbEngineMoves').append('Bilgisayar');
27              $('#form').hide();
28              $('#wrapper').show();
29              if(mode == 1){
30                  GameBoard.side = 1^playerSide;
31                  PreSearch();
32                  GameBoard.side = playerSide;
33                  $('#Timer').hide();
34                  $('#TimerOpp').hide();
35              }else if(mode == 2){
36                  $('#TimerUserName').append('<b>('+userName+'):</b>');
37                  $('#TimerUserNameOpp').append('<b>(Bilgisayar):</b>');
38                  TotalSeconds = timing * 60;
39                  TotalSecondsOpp = timing*60;
```

```
40              Increments = increment;
41              $('#ThinkingTime').hide();
42              $('#TimeInfo').show();
43              if(timing == 0){
44                $('#TimeInfo').append('Sınırsız');
45                $('#Timer').hide();
46                $('#TimerOpp').hide();
47              }else{
48                  if(increment == 0){
49                      $('#TimeInfo').append(timing  +  ' dk.');
50                  }else{
51                      $('#TimeInfo').append(timing  +  ' dk. ve ' +
                        increment + ' sn. artışlı');
52                  }
53                  $('#Timer').show();
54                  $('#TimerOpp').show();
55                  TimerOn = BOOL.TRUE;
56                  TimerOnOpp = BOOL.TRUE;
57                  window.clearTimeout(setClearTime);
58                  Tick();//Sayaç başlatılıyor.
59              }
60
61              SearchController.best = NOMOVE;
62              GameBoard.side = 1^playerSide;
63              PreSearch();
64              if (SearchController.best != NOMOVE){
65                  TotalSecondsOpp += parseInt(Increments);
66                  CountDownTimerOpp.innerHTML =
                    UpdateTimer(TotalSeconds);
67                  TimerOnOpp = BOOL.FALSE;
68              }
69              GameBoard.side = playerSide;
70          }
71      }
72
73      if(userName != '' & player == 2 & playerSide != ''){
74
75          socket.emit('Two Player Setup',
            {name: userName, player:player, side:playerSide,
            timing:timing, increment:increment});
76          $('#opponentWait').show();
77          $('#form').hide();
78          $('#wrapper').hide();
79
```

```
80          }
81      });
```

15. satırda, "Başlat" düğmesine basıldığında (selectionForm submit) bu satırdaki anonim form başlatılmaktadır. 17. satırda, sayfanın kendini yenilemesini engellemek için varsayılan olarak yapılması gereken işlemler engellenmektedir. 18. satırda bir kullanıcı ismi olup olmadığı ve oyuncu sayısının "1" olup olmadığı kontrol edilmektedir. Öyleyse 19. satırda, oyuncunun tarafının siyah olup olmadığı kontrol edilmektedir. Öyleyse yani, tek oyunculu ve tarafın siyah olduğu durumlarda 20. satırda oyun tahtasının 180 derece çevrilmesi gerektiğini belirten değişkene BOOL.TRUE değeri atanmaktadır. Böylelikle oyuncu ekranın alt tarafında seçtiği tarafın taşlarını görebilmektedir. 21. satırda, **SetInitialBoardPieces** fonksiyonu çağrılarak GameBoard.pieces dizisinde tutulan dizi elemanları oyun tahtasında kendi yerlerine yerleştirilmektedir. 23. satırda, rakip bilgisayar olduğu için side1 ile tanımlanan alana "Bilgisayar" yazılmaktadır. 24. satırda ise side0 alanına userName değişkenindeki değer (kullanıcının kendisi için belirlemiş olduğu isim) yazdırılmaktadır. 25. satırda oyuncunun hamlelerini gösteren alanın üzerine userName değişkenindeki değer yazdırılmaktadır. 26. satırda ise rakibin hamlelerinin yazdırılacağı alanın üzerine rakibin ismi yani, şu anki durumda "Bilgisayar" yazdırılmaktadır. 27. satırda, "form" tanımlamasına sahip olan alan yani, oyun kurulumu için seçimlerin yapıldığı alan artık ihtiyaç kalmadığı için ve artık oyunun başlatılması gerektiği için gizlenmektedir. 28. satırda, oyunun oynanabilmesi için gerekli alanların içerisinde yer aldığı alan olan "wrapper" ismiyle tanımlanmış olan alan görünür duruma getirilmektedir. Şayet oyun modu seçilmişse (29. satır) 30. satırda GameBoard.side değişkenine rakibin tarafı ne ise o değer atanmaktadır. 31. satırda da **PreSearch** fonksiyonu çağrılarak rakip için en iyi hamlenin bulunması sağlanmaktadır. Rakip bilgisayar olduğu için bilgisayar kendisi için en iyi hamleyi bulmaya çalışmaktadır. 32. satırda GameBoard.side değişkenine oyuncunun tarafı atanmaktadır. Böylelikle oyuncunun hamlesini yapabileceği ortam oluşturulmaktadır. Çünkü bir taraf için geçerli hamleler **generateMoves** fonksiyonu tarafından GameBoard.side değişkeninin değerine bakılarak oluşturulmaktadır. Şayet bu değer yanlışsa geçerli hamleler karşı taraf için oluştulmaktadır ve asıl hamle yapması gereken taraf için geçerli hamle bulunmamaktadır. 33. ve 34. satırlarda sayaçların yer aldığı alanlar gizlenmektedir. Çünkü öğrenme modunda sayaçlar devrede değildir. Şayet oyun modu (mode == 2) seçeneği seçilmişse (35. satır) bu sefer sayaçlar devrede olacaktır. Bu nedenler hangi sayacın kimin olduğunun belli olması için oyuncu isimleri sayaçların olduğu kısma yazılmaktadır (36. ve 37. satırlar). 38. ve 39.

satırlarda, belirlenen süre her iki tarafın sayacı tutan değişkenlerine saniye cinsine çevrilerek atanmaktadır. 40. satırda, artış miktarı Increments değişkenine aktarılmaktadır. Her iki taraf için kullanılacak artış miktarı aynı olduğu için tek bir değişken tutulmaktadır. Oyun modunda bilgisayarın düşünme süresi değiştirilmediğinden dolayı o alan 41. satırda gizlenmektedir. 42. satırda, sayaçların bulunduğu alan görünür duruma getirilmektedir. 43. satırda, seçilmiş olan oyun süresi değerinin "0" yani, sınırsız olup olmadığı değerlendirilmektedir. Öyle ise 44. satırda Oyun süresi bilgisini veren alana "Sınırsız" yazdırılmaktadır. Bu durumda sayaç kullanılmayacağı için 45. ve 46. satırda, sayaçların bulunduğu alanlar gizlenmektedir. Şayet bir oyun süresi belirlenmişse 48. satırda bir artış değeri belirlenip belirlenmediği kontrol edilmektedir. Şayet bir artış süresi belirlenmediyse 49. satırda oyun süresinin yanına "dk." eklenerek oyun süresi alanına yazdırılmaktadır. Şayet bir artırım miktarı belirlenmişse bu sefer 51. satırda bu bilgi ve oyun süresi bilgisi ilgili alana yazdırılmaktadır. 53. ve 54. satırlarda sayaçlar görünür duruma getirilmektedir. 55. ve 56. satırlarda, sayaçların çalıştırılması ve durdurulması için kullanılan değişkenlere sayaçların çalıştırılması için BOOL.TRUE değeri atanmaktadır. 58. satırda da **Tick** fonksiyonu (Bkz. 4.3) çağrılarak sayaçlar başlatılmaktadır. 61. satırda, bilgisayarın en iyi hamlesinin tutulduğu değişkeninin içeriği sıfırlanmaktadır. Bunun nedeni 63. satırda yapılacak olan en iyi hamle araması sonucunda geçerli bir hamle bulunamazsa 64. satırda yapılan kontrolde eskiden kaydedilmiş bir hamleye göre karar verilmemesi içindir. 62. satırda, GameBoard.side değişkenine bilgisayarın tarafı atanmakta ve 63. satırda o taraf için en iyi hamle aranmaktadır. **PreSearch** fonksiyonu SearchController.best değişkenini güncellemektedir. 64. satırda bir hamle bulunup bulunmadığı kontrol edilmektedir. Şayet bulunmuşsa 65. satırda, geçerli hamle başına yapılacak olan artış miktarı bilgisayarın oyun süresine eklenmektedir. 66. satırda, yeni oyun süresi grafik kullanıcı ara yüzünde güncellenmektedir. 67. satırda, bilgisayarın sayacını kontrol eden TimerOnOpp değişkenine BOOL.FALSE değeri atanarak bilgisayarın sayacı durdurulmaktadır. 69. satırda, oyuncunun hamle yapmasını sağlamak için GameBoard.side değişkenine oyuncunun taraf bilgisi atanmaktadır. Şayet ağ üzerinde iki oyunculu bir oyunsa, bir isim belirlenmişse ve oyuncu tarafı da belirlenmişse 73. satırdaki kontrol sonucunda 75. satıra geçilmektedir. İki oyunculu bir oyun olduğu için oyuncu tarafında belirlenen kriterlere ait bilgilerin sunucuya gönderilmesi ve bu bilgilerin oyuncunun kendisi haricindeki tüm kullanıcılara gönderilmesi gerekmektedir. 75. satırda belirlenen kriterler sunucuya gönderilmektedir. Diğer kullanıcılara gönderme işlemi sunucu tarafından halledilmektedir. 76. satırda, yeni bir oyun kurulduğuna ve bir rakip beklendiğine dair bir mesajın gösterildiği alan

görünür duruma getirilmektedir. Oyun kriterleri belirlendiği için "form" alanı gizlenmektedir (77. satır). Oyuna bir rakip oyuncu dahil olana kadar da "wrapper" alanı gizlenmektedir. Çünkü karşıda oyun oynayacak bir rakip henüz bulunmamaktadır.

Sunucu "Two Player Setup" etiketli bir mesaj aldığında server.js içerisindeki aşağıdaki kodlar çalıştırılmaktadır;

```
48      socket.on('Two Player Setup', function(data){
49
50          if (data.side == COLOURS.WHITE) {
51              socket.side = 0;
52          }else{
53              socket.side = 1;
54          }
55
56          waitingList.push(
57              {
58                  name: data.name,
59                  side: data.side,
60                  timing:data.timing,
61                  increment:data.increment
62              });
63          io.sockets.emit('waitingList', waitingList);
64      });
```

48. satırda, "Two Player Setup" etiketli bir mesaj alındığında o satırdaki anonim fonksiyon data parametresi ile başlatılmaktadır. "data" parametresi istemci tarafından gönderilen veri setidir. 50. satırda, alınan data parametresindeki "side" bilgisinin beyaz olup olmadığı kontrol edilmektedir. Şayet beyazsa istemcinin soket bilgisine bu değer "socket.side = 0" (51. satır) şeklinde atanmaktadır. Değilse "socket.side = 1" (53. satır) şeklinde atanmaktadır. Böylelikle o istemciye ait soket numarasına hem oyuncunun ismi (server.js 41. satır) hem de taraf bilgisi (51. ve 53. satırlar) bağlanmış olmaktadır. 56. ve 62. satırlar arasında, data parametresi ile alınan bilgiler "waitingList" dizisine veri yapısı şeklinde son eleman olarak eklenmektedir. 63. satırda, güncellenmiş olan "waitingList" dizisi tüm istemcilere gönderilmektedir. İstemci tarafından "waitingList" etiketli bir mesaj alındığında aşağıdaki işlemler yapılmaktadır (index.js dosyası);

```
176     socket.on('waitingList', function(data){
```

```
177       wListLength = data.length;
178       if (wListLength == 0){
179          $('.waitingListWrapper').hide();
180       }else if(wListLength > 0){
181          $('.waitingList').html('<h3>İsim : </h3>');
182          $('.waitingListSide').html('<h3>Taraf : </h3>');
183          $('.waitingListTiming').html('<h3>Süre : </h3>');
184
185          var wListHtml ='<h3>İsim : </h3>';
186          var wListSideHtml ='<h3>Taraf : </h3>';
187          var wListTimingHtml ='<h3>Süre : </h3>';
188
189          for(var i=0; i < data.length; i++ ){
190            wListHtml += '<p>' + data[i].name + '</p>';
191            wListSideHtml += '<p>' + SideColourString[data[i].side] + '</p>';
192            if( data[i].timing == 0 ){
193                wListTimingHtml += '<p>' + "Sınırsız" + '</p>';
194            }else{
195                wListTimingHtml += '<p>' + data[i].timing + 'dk.' + ' + ' +
                    data[i].increment + 'sn.' + '</p>';
196            }
197          }
198
199          $('.waitingList').html(wListHtml);
200          $('.waitingListSide').html(wListSideHtml);
201          $('.waitingListTiming').html(wListTimingHtml);
202          if(player == 2)     $('.waitingListWrapper').show();
203          $(".waitingList p").click(function() {
204               opponentName = $( this ).html();
205               socket.emit('opponent selected', opponentName );
206          });
207       }
208    });
```

176. satırda, "waitingList" etiketli bir mesaj alındığında bu satırdaki anonim fonksiyon, alınan mesajla birlikte gönderilmiş olan data (waitingList dizisi) parametresi kullanılarak başlatılmaktadır. 177. satırda, gönderilmiş olan dizinin uzunluğu wListLength değişkenine atanmaktadır. Şayet bekleme listesinde herhangi bir eleman yoksa (178. satır) bekleme listesinin gösterildiği alan gizlenmektedir (179. satır). Şayet bekleme listesinde eleman varsa (180. satır) listedeki elemanların "İsim", "Taraf" ve "Süre" bilgileri html formatına uygun halde listelenerek sırasıyla wListHtml, wListSideHtml ve wListTimingHtml

değişkenlerine atanmaktadır (181. ve 197. satırlar arası). 199. ve 201. satırlar arasında bu değişkenler index.ejs dosyası içerisinde yer alan uygun yerlere yazdırılarak rakip bekleyen oyuncu listesi güncellenmektedir. 202. satırda, şayet oyun iki oyunculuysa bu liste görünür hale getirilmektedir. 203 satırda, "waitingList" isimli sınıfın p (paragraf) elemanlarından birisi tıklandığında yani, rakip bekleyen oyunculardan birisi tıklandığında bu satırdaki anonim fonksiyon çalıştırılmaktadır. Seçilen isim, seçimi yapan oyuncunun rakibi olacağından dolayı rakip olarak seçilen oyuncunun ismi opponentName değişkenine atanmaktadır (204). 205. satırda, bu isim "opponent selected" etiketi kullanılarak sunucuya gönderilmektedir. Sunucu tarafından "opponent selected" etiketli bir mesaj alındığında server.js içerisindeki aşağıdaki kod çalıştırılmaktadır;

```
66  socket.on('opponent selected', function(opponentName){
67    for (var i = 0 ; i < waitingList.length ; i++){
68      if(opponentName == waitingList[i].name){
69        socket.opponent = allPlayers[opponentName];
70        allPlayers[opponentName].emit('challenge accepted',
          {nickname:socket.nickname, timing:waitingList[i].timing,
          increment:waitingList[i].increment});
71        waitingList.splice(i, 1);
72        io.sockets.emit('waitingList', waitingList);
73        break;
74      }else if(i == waitingList.length){
75        io.sockets.emit('waitingList', waitingList);
76        allPlayers[socket.nickname].emit('no opponent', opponentName);
77      }
78    }
79  });
```

66. satırda, "opponent selected" etiketli bir mesaj alındığında bu mesajla birlikte gönderilen opponentName bu satırdaki anonim fonksiyona parametre olarak verilerek bu fonksiyon çalıştırılmaktadır. 67. ve 78. satırlar arasında, waitingList dizisini gezinmek için bir for döngüsü kullanılmaktadır. Bu döngü içerisinde bir if sorgusuyla rakibin hala rakip bekleyen oyuncular içerisinde olup olmadığı kontrol edilmektedir (68. satır). Çünkü - her ne kadar çok küçük bir ihtimal de olsa - istemci tarafındaki listeler güncellenmeden iki farklı oyuncu tarafından aynı kişinin rakip olarak seçilme ihtimali vardır. Şayet seçilen rakip hala "waitingList" içerisinde ise henüz başka biri tarafından seçilmemiş demektir ve iki oyuncunun eşleşmesi için seçimi yapan oyuncunun soket numarasına rakip olarak belirlenen oyuncunun soket numarası bağlanmaktadır (69. satır).

socket.opponent değişkeninin "socket" kısmında seçimi yapan oyuncunun soket numarası, ".opponent" kısmı da seçilen rakibin soket numarası tutulmaktadır. 70. satırda, rakip olarak seçilen oyuncuya o oyuncunun soket numarası (allPlayers[opponentName]) kullanılarak "challenge accepted" etiketi ile seçimi yapan oyuncunun nickname, timing ve increment bilgilerini içeren bir mesaj gönderilmektedir. Böylelikle seçilen oyuncuya kimin tarafından seçildiği bilgisi iletilmiş olmaktadır. Seçilmiş ve uygun şekilde eşleşmesi tamamlanmış olan rakibin artık waitingList'te olmaması gerektiğinden dolayı 71. satırda listeden çıkarılmaktadır. 72. satırda, yeni waitingList yayınlanmaktadır. 73. satırda, gerekli işlem yapıldığı için for döngüsünden çıkılmaktadır. Şayet seçilmiş olan rakip listede bulunamazsa bu iki farklı kişinin aynı rakibi birbirine çok yakın zamanda seçtiği anlamına gelmektedir. Bu durumda seçimi yapan oyuncunun waitingList'inin güncellenmesi için "waitingList" tekrar gönderilmektedir (75. satır). Bir de rakibin listede bulunamadığını gösteren "no opponent" mesajı gönderilmektedir (76. satır). Rakip olarak seçilen oyuncuya "challenge accepted" mesajı gönderildiğince aşağıdaki şekilde işlem yapılmaktadır (index.js);

```
210  socket.on('challenge accepted', function(opponent){
211      socket.emit('challenge confirmed',
         {nickname:opponent.nickname, timing:opponent.timing,
         increment:opponent.increment} );
212  });
```

Rakip olarak seçilen oyuncu tarafından "challenge accepted" mesajı alındığında sunucu tarafından gönderilmiş olan ve karşı tarafın bilgilerini içeren opponent parametresiyle anonim bir fonksiyon başlatılmaktadır. 211. satırda, sunucuya "challenge confirmed" etiketliyle, sunucudan alınan rakip bilgilerini aynı şekilde hiç değiştirmeden gönderilmektedir. Bunun amacı oyun teklifini yapan oyuncunun hala ağa bağlı ve oyun için müsait olduğunun bildirilmesidir. Sunucu bu mesajı aldığında aşağıdaki işlemleri yapmaktadır (server.js);

```
89   socket.on('challenge confirmed', function(opponent){
90       socket.opponent = allPlayers[opponent.nickname];
91       allPlayers[opponent.nickname].emit('start game', {
         yourName: opponent.nickname, yourSide: socket.side^1,
         opponentName: socket.nickname, opponentSide:socket.side,
         timing:opponent.timing, increment:opponent.increment});
92       allPlayers[socket.nickname].emit('start game', {
         yourName: socket.nickname, yourSide: socket.side ,
         opponentName: opponent.nickname, opponentSide: socket.side^1,
         timing:opponent.timing, increment:opponent.increment});
```

```
93      });
```

Listeden seçilen oyuncu tarafından gönderilen "challenge confirmed" mesajı sunucu tarafından alındığında, gelen mesajla birlikte gelen ve rakibin bilgilerini içeren (seçimi yapan oyuncu) opponent parametresi ile anonim bir fonksiyon başlatılmaktadır (89. satır). 90. satırda, seçilen oyuncunun soket numarasına seçen oyuncunun soket numarası bağlanmaktadır. 91. satırda, seçen oyuncuya, 92. satırlarda ise seçilen oyuncuya "start game" etiketiyle, kendi isimleri, kendi tarafları, rakibin ismi, rakibin tarafı, oyun süresi ve artış miktarı bilgileri gönderilmektedir. Bu noktada her iki oyuncu da hem kendisinin hem de rakibin bilgilerini alarak oyuna başlamaya hazır duruma gelmektedir. Oyuncular (istemciler) "start game" etiketi ile bir mesaj aldığında aşağıdaki işlemler yapılmaktadır;

```
214     socket.on('start game', function(data){
215       player = 2;
216       opponentName = data.opponentName;
217       $('#Chat').show();
218       $('#TimerUserName').append('<b>('+data.yourName+'):</b>');
219       $('#TimerUserNameOpp').append('<b>('+data.opponentName+'):</b>');
220
221       if( data.yourSide == COLOURS.BLACK){
222           GameController.BoardFlipped = BOOL.TRUE;
223           SetInitialBoardPieces();
224       }
225
226       GameBoard.side = data.yourSide;
227       playerSide = data.yourSide;
228       opponentSide = data.yourSide^1;
229       if(   playerSide == 0 ){
230           $('#side0').append(data.yourName);
231           $('#side1').append(data.opponentName);
232
233           $('#lbUserMoves').append(data.yourName);
234           $('#lbEngineMoves').append(data.opponentName);
235       }else{
236           $('#side0').append( data.yourName );
237           $('#side1').append( data.opponentName );
238
239           $('#lbUserMoves').append( data.yourName );
240           $('#lbEngineMoves').append( data.opponentName );
```

```
241         }
242
243         $('.waitingListWrapper').hide();
244         $('#opponentWait').hide();
245         $('#form').hide();
246         $('#FenInDiv').hide();
247
248         if(data.timing == 0){
249             $('#TimeInfo').append('Sınırsız');
250             $('#Timer').hide();
251             $('#TimerOpp').hide();
252         }else{
253             if(data.increment == 0){
254                 $('#TimeInfo').append(data.timing  +  ' dk.');
255             }else{
256                 $('#TimeInfo').append
                    (data.timing  +  ' dk. ve ' + data.increment + ' sn. artışlı');
257             }
258         }
259
260         $('#ThinkingTime').hide();
261         $('#TimeInfo').show();
262
263         $('#wrapper').show();
264
265         timing = data.timing;
266         increment = data.increment;
267         TotalSeconds = data.timing * 60;
268         TotalSecondsOpp = data.timing * 60;
269         Increments = data.increment;
270         if(TotalSeconds > 0){
271             Tick();//Sayaç başlatılıyor.
272         }
273     });
```

214. satırda, "start game" etiketli bir mesaj alındığında, mesajla birlikte gönderilmiş olan "data" parametre olarak kullanılarak bir anonim fonksiyon başlatılmaktadır. "data" içerisinde data.yourName, data.yourSide, data.opponentName, data.opponentSide, data.timing ve data.increment bilgileri yer almaktadır. İstemci tarafından mesajları doğru oyuncuya göndermek için soket numaralarının bilinmesine gerek yoktur. Çünkü soket bilgisi sunucu tarafından tutulmaktadır. İstemciler gönderecekleri mesajı rakibin ismine gönderdiğinde

sunucu aldığı mesajı o rakibe ait soket numarasına yönlendirmektedir. 215. satırda, oyun iki kişilik olduğu için bu değer player değişkenine atanmaktadır. 216. satırda rakibin ismi opponentName değişkenine atanmaktadır. 217. satırda rakiplerin birbiriyle mesajlaşması için "Chat" ile tanımlanan alan görünür duruma getirilmektedir. 218. ve 219. satırlarda sayaçların tutulduğu alana oyuncu ve rakibinin isimleri yazdırılmaktadır. Şayet oyuncunun tarafı siyahsa (221. satır) GameController.BoardFlipped değişkeni BOOL.TRUE yapılmaktadır (222. satır) ve grafik kullanıcı ara yüzündeki oyun taşları tekrar tahtaya yerleştirilmektedir (223. satır). Bunun amacı siyah tarafın taşlarını bilgisayar ekranında alt tarafa getirerek oyuncunun oyunu rahat oynamasını sağlamaktır. Bu işlem beyaz taraf için burada yapılmamaktadır. Çünkü oyun kurulumunda varsayılan dizilim olarak beyaz taraf kullanıcıya yakın olarak yerleştirilmektedir. 226. ve 228. satırlar arasında taraf bilgileri uygun değişkenlere atanmaktadır. 229. ve 241. satırlar arasında taraf bilgileri kontrol edilerek oyuncunun ve rakibinin isimleri uygun alanlara yazdırılmaktadır. 243. ve 246. satırlar arasında, oyun kurulumu esnasında kullanılan alanlara artık ihtiyaç kalmadığı için gizlenmektedir. Şayet oyun süresi belirlenmemişse (248. satır) oyun süresi ile ilgili alana "Sınırsız" yazdırılmakta (249. satır) ve sayaçların bulunduğu alanlar gizlenmektedir (250. ve 251. satırlar). Oyun süresi belirlenmişse (252. satır) bu sefer artış miktarı belirlenip belirlenmediğine bakılmaktadır (253. satır). Artış miktarı belirlenmemişse sadece oyun süresi bilgisi ekrana yazdırılmaktadır (254. satır). Artış miktarı belirlenmişse (255. satır) oyun süresi bilgisinin yanında artış miktarı bilgisi de ekrana yazdırılmaktadır (256. satır). 260. satırda, bilgisayarın düşünme süresi kullanılmadığından dolayı gizlenmektedir. 261. satırda, oyun süresi bilgisini gösteren alan görünür hale getirilmektedir. 263. satırda "wrapper" olarak tanımlanan alan görünür duruma getirilerek oyun tahtasının ve yardımcı alanların ekranda görünmesi sağlanmaktadır. 265. ve 266. satırlarda, oyun süresi bilgisi ve artış miktarı değişkenlere atanmaktadır. Bu değişkenler aynı oyun tekrar oynanmak istendiğinde kullanılmaktadır. 267. ve 268. satırlarda, oyuncu ve rakibinin toplam oyun süreleri saniye cinsinden değişkenlere atanmaktadır. 269. satırda artış miktarı bir değişkene atanmaktadır. Artık oyuncuların ekranında oyuna başlamaları için gerekli her şey hazır olduğu için sayaçlar başlatılması gerekmektedir. 270. satırda, oyun süresinin "0" büyük olup olmadığı kontrol edilerek sayaçların geriye doğru saymaya başlaması için 271. satırda **Tick** (Bkz.3.25) fonksiyonu çalıştırılmaktadır.

Oyun esnasında "Oyundan Çekil" düğmesine basarak oyuncular oyundan çekilebilmektedirler. Bu durumdan rakibin haberdar edilmesi gerekmektedir.

Bunun için "Oyundan Çekil" düğmesine basıldığında aşağıdaki işlemler yapılmaktadır (gui.js);

```
1  $('#Resign').click(function(){
2      GameController.GameOver = BOOL.TRUE;
3      TimerOn = BOOL.FALSE;
4      TimerOnOpp = BOOL.FALSE;
5      $('#NewGameButton').attr("disabled", false);
6      $('#Resign').attr("disabled", true);
7      if(playerSide == COLOURS.WHITE & player == 1){
8          $("#GameStatus").text("OYUN BİTTİ
           {Siyah yendi - Oyundan çekildiniz}");
9      }else if(playerSide == COLOURS.BLACK & player == 1){
10         $("#GameStatus").text("OYUN BİTTİ
           {Beyaz yendi - Oyundan çekildiniz}");
11     }else if(playerSide == COLOURS.WHITE & player == 2){
12         $("#GameStatus").text("OYUN BİTTİ
           {Siyah yendi - Oyundan çekildiniz}");
13         socket.emit('Resign', playerSide);
14     }else if(playerSide == COLOURS.BLACK & player == 2){
15         $("#GameStatus").text("OYUN BİTTİ {Beyaz yendi -
           Oyundan çekildiniz}");
16         socket.emit('Resign');
17     }
18 });
```

İlk satırda, "Oyundan Çekil" düğmesine basıldığında anonim bir fonksiyon başlatılmaktadır. 2. satırda, oyunun bitip bitmediğini kontrol eden değişken BOOL.TRUE yapılmaktadır. Artık sayaçlara ihtiyaç kalmadığı için onlar da BOOL.FALSE yapılmaktadır (3.ve 4. satırlar). Oyuncunun tekrar oynamak isteyeceği düşünülerek 5. satırda "Tekrar Oyna" butonu aktif hale getirilmektedir. Hali hazırda oyundan çekilme gerçekleşmiş olduğu için 6. satırda "Oyundan Çekil" düğmesi pasif yapılmaktadır. 7. ve 17. satırlar arasında, oyundan çekilen taraf tespit edilerek hangi tarafın oyunu kazandığını gösteren bir mesaj yazdırılmaktadır. Aynı zamanda sunucuya "Resign" etiketli ve oyundan çekilen oyuncunun hangi tarafta olduğunu gösteren bir mesaj gönderilmektedir. Sunucu tarafından alınan "Resign" etiketli mesaj alındığında aşağıda gösterildiği gibi o oyuncunun rakibine "Opponent Resigned" etiketli bir mesaj gönderilmektedir (server.js - 116. ve 118. satırlar);

```
116  socket.on('Resign', function(){
117      socket.opponent.emit('Opponent Resigned');
```

```
118   });
```

Rakip oyuncu tarafından "Opponent Resigned" mesajı alındığında aşağıdaki şekilde işlem yapılmaktadır (index.js);

```
275   socket.on('Opponent Resigned', function(){
276     GameController.GameOver = BOOL.TRUE;
277     TimerOn = BOOL.FALSE;
278     $('#NewGameButton').attr("disabled", false);
279     $('#Resign').attr("disabled", true);
280     if( playerSide == 0 ){
281       $("#GameStatus").text("OYUN BİTTİ
          {Beyaz yendi - Rakibiniz oyundan çekildi}");
282     }else if( playerSide == 1 ){
283       $("#GameStatus").text("OYUN BİTTİ
          {Siyah yendi - Rakibiniz oyundan çekildi}");
284     }
285   });
```

275. satırda, "Opponent Resigned" etiketli mesaj alındığında bir anonim fonksiyon başlatılmaktadır. 276. satırda, oyunun bittiği GameController.GameOver değişkenine BOOL.TRUE değeri atanarak belirtilmektedir. 277. satırda oyuncunun sayacı durdurulmaktadır. 278. ve 279. satırlarda duruma uygun olarak düğmeler aktif/pasif konumlarına alınmaktadır. 280. ve 284. satırlar arasında, oyunun bittiği ve yenen tarafın kim olduğuna dair bir mesaj yazdırılmaktadır. Oyun bir şekilde sonlandığında oyuncular aynı oyunu tekrar oynamak isteyebilmektedirler. Bunun için "Tekrar Oyna" butonu kullanılmaktadır. Bu durumda gui.js dosyasında aşağıdaki kodlar çalıştırılmaktadır;

```
1  $('#NewGameButton').click( function () {
2
3      if(player == 1){
4
5        SearchController.best = NOMOVE;
6        NewGame(START_FEN);
7
8        if (mode == 2){
9          window.clearTimeout(setClearTime);
10         TimerOn = BOOL.TRUE;
11         TimerOnOpp = BOOL.TRUE;
```

```
12              $('#NewGameButton').attr("disabled", true);
13              $('#Resign').attr("disabled", false);
14              TotalSeconds = timing * 60;
15              TotalSecondsOpp = timing * 60;
16              Tick();
17              if (SearchController.best != NOMOVE){
18                  TotalSecondsOpp += parseInt(Increments);
19                  CountDownTimerOpp.innerHTML = UpdateTimer(TotalSeconds);
20                  TimerOnOpp = BOOL.FALSE;
21              }
22          }
23      }else if(player == 2){
24              socket.emit('New Game Request');
25      }
26      noForward();
27  });
```

"Tekrar Oyna" düğmesine basıldığında oyunun tek oyunculu mu iki oyunculu mu olduğu kontrol edilmektedir (3. ve 23. satırlar). Tek oyunculuysa 5. ve 22. satırlar arasındaki kodlar çalıştırılmaktadır. Bu kısımdaki kodlar tek oyunculu oyun başlatma kısmındaki kodlara oldukça benzer kodlardır. Şayet oyun iki oyunculuysa sunucuya "New Game Request" etiketli bir mesaj gönderilmektedir (24. satır). Ayrıca 26. satırda **noForward** fonksiyonu çağrılmaktadır. Bu fonksiyon yardımıyla, mevcut hisPly'den ötesi sıfırlanmaktadır. Aksi takdirde hamle yapıldıktan sonra bile -şayet oyun daha önce geri alınmışsa veya bir önceki oyunun bilgileri silinmemişse- ileri butonuna basılarak ileri hamle yapılabilmektedir. Bu kod makemove.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function noForward(){
2      var index;
3      for(index = GameBoard.hisPly ; index < MAXGAMEMOVES; ++index) {
4          GameBoard.history[index].playerMove = NOMOVE;
5          GameBoard.history[index].opponentMove = NOMOVE;
6          GameBoard.history[index].castlePerm = 0;
7          GameBoard.history[index].wEnPas = SQUARES.NO_SQ;
8          GameBoard.history[index].bEnPas = SQUARES.NO_SQ;
9          GameBoard.history[index].fiftyMove = 0;
10         GameBoard.history[index].posKey = 0;
11     }
12 }
```

"New Game Request" etiketli mesaj sunucu tarafından alındıktan sonra aşağıdaki gibi aynı etiketle rakibe iletilmektedir;

```
120    socket.on('New Game Request', function(){
121        socket.opponent.emit('New Game Request');
122    });
```

Rakip oyuncu bu mesajı aldığında aşağıdaki şekilde işleme almaktadır(index.js);

```
287   socket.on('New Game Request',function(){
288     if(confirm ('Rakibinizin tekrar oynama isteğini kabul ediyor musunuz?')){
289         socket.emit('New Game Request Accepted');
290         DeselectSq(UserMove.from);
291         DeselectSq(UserMove.to);
292         NewGame(START_FEN);
293         $("#GameStatus").text("");
294         $('#NewGameButton').attr("disabled", true);
295         $('#Resign').attr("disabled", false);
296         window.clearTimeout(setClearTime);
297         TimerOn = BOOL.TRUE;
298         TimerOnOpp = BOOL.TRUE;
299         TotalSeconds = timing*60;
300         TotalSecondsOpp = timing*60;
301         Increments = increment;
302         Tick();
303     }else{
304         socket.emit('New Game Request Rejected');
305     }
306   });
```

287. satırda, alınan mesaj için bir anonim fonksiyon başlatılmaktadır. 288. satırda, kullanıcının ekranında oyunu tekrar oynamak isteyip istemediğine dair bir mesaj çıkarılmaktadır. Şayet kullanıcı bu mesaja olumlu yanıt verirse 289. satırdan devam edilmektedir. 288. satırda, yeniden oynama isteğinin kabul edildiğini belirten "New Game Request Accepted" mesajı sunucuya gönderilmektedir. 290. ve 291. satırlarda, oyuncunun yaptığı hamle neticesinde seçilmiş olarak gösterilen karelerin seçim işaretleri kaldırılmaktadır. Bir hamle yapıldığında karşı taraf henüz hamle yapmadan oyundan çekilip yeni oyun isteğinde bulunduğu zaman hali hazırda yapılmış olan hamle seçimi silinmediğinden dolayı bu iki satır o seçilen karelerin seçim işaretlerini silmek

için kullanılmaktadır. 292. satırda, **NewGame** fonksiyonu çağrılarak oyun ekranı ve diğer değişkenler sıfırlanmaktadır. 293. satırda, oyun sonucunu belirten alan sıfırlanmaktadır. 294. ve 295. satırlarda, "Tekrar Oyna" ve "Oyundan Çekil" düğmeleri pasif yapılmaktadır. 296. satırda, hali hazırda çalışan sayaç durdurulmaktadır. 297. ve 301. satırlar arasında, sayaçla ilgili değerler güncellenmektedir. 302. satırda, sayaçlar başlatılmaktadır. Şayet yeni oyun isteği kabul edilmezse karşı tarafa "New Game Request Rejected" etiketli bir mesaj gönderilmektedir (304. satır). Aşağıda da görüldüğü gibi oyun isteğinin kabul edildiğine veya reddedildiğine dair gönderilen mesajlar sunucu tarafından hiç değiştirilmeden karşı tarafa iletilmektedir (server.js - 124. ve 130. satırlar arası).

```
124  socket.on('New Game Request Accepted', function(){
125    socket.opponent.emit('New Game Request Accepted');
126  });
127
128  socket.on('New Game Request Rejected', function(){
129    socket.opponent.emit('New Game Request Rejected');
130  });
```

Yeniden oynama isteği yapan taraf tarafından "New Game Request Accepted" etiketli mesaj alındığında aşağıdaki işlemler yapılmaktadır (index.js).

```
308    socket.on('New Game Request Accepted', function(){
309        DeselectSq(UserMove.from);
310        DeselectSq(UserMove.to);
311        NewGame(START_FEN);
312        $("#GameStatus").text("");
313        $('#NewGameButton').attr("disabled", true);
314        $('#Resign').attr("disabled", false);
315        window.clearTimeout(setClearTime);
316        TimerOn = BOOL.TRUE;
317        TimerOnOpp = BOOL.TRUE;
318        TotalSeconds = timing*60;
319        TotalSecondsOpp = timing*60;
320        Increments = increment;
321        Tick();
322    });
```

Yukarıdaki işlemler oyunu yeniden oynamayı kabul eden tarafın yaptığı işlemlerle aynı işlemlerdir. Şayet "New Game Request Rejected" mesajı alınırsa,

aşağıda görüldüğü gibi bir mesajla bu isteği yapan tarafın ekranına bir uyarı olarak yazdırılmaktadır (index.js);

```
324     socket.on('New Game Request Rejected', function(){
325         alert('Tekrar oynama isteğiniz kabul edilmedi!');
326     });
```

Oyun esnasında rakip Internet gezginini kapatarak da oyunu terk edebilmektedir. Bu durumlarda istemci tarafından otomatik olarak "disconnect" etiketli bir mesaj gönderildiği için bu durum sunucu tarafından tespit edilebilmektedir. Sunucu tarafından "disconnect" etiketli mesaj alındığında aşağıdaki işlemler yapılmaktadır (server.js);

```
 96 socket.on('disconnect', function(data){
 97
 98     if(!socket.nickname)return;
 99     if ( socket.opponent != null ){
100         socket.opponent.emit('Disconneted', socket.nickname);
101     }
102
103     console.log(socket.nickname + ' DISCONNECTED');
104     delete allPlayers[socket.nickname];
105
106     for (var i = 0 ; i < waitingList.length ; i++){
107         if(socket.nickname == waitingList[i].name){
108             waitingList.splice(i, 1);
109             io.sockets.emit('waitingList', waitingList);
110             break;
111         }
112     }
113
114 });
```

96. satırda, bir anonim fonksiyon başlatılmaktadır. 98. satırda, mesajın alındığı soket numarasına bağlı bir nickname (isim) olup olmadığına bakılmaktadır. Şayet bir isim yoksa daha fazla işlem yapılmadan fonksiyondan çıkılmaktadır. Çünkü istemci sadece Internet gezgininde eş zamanlı satranç oyununu web sitesini açmış ve hiçbir işlem yapmadan kapatmış olabilme ihtimali mevcuttur. Şayet bir isim varsa 99. satırdan devam edilmektedir. 99. satırda bu sefer o soket numarasına bağlı bir rakip olup olmadığı kontrol edilmektedir. Bir rakip mevcutsa o rakibe "Disconnected" etiketli bir mesaj gönderilmektedir (100.

satır). 103. satırda oyuncunun oyundan ayrıldığı konsol ekranına yazdırılmaktadır. 104. satırda, oyuncu allPlayer dizisinden silinmektedir. 106. ve112. satırlar arasında, bir for döngüsüyle waitingList dizisi gezilerek oyuncu listeden çıkarılmakta ve tüm kullanıcılara yeni liste gönderilmektedir. "Disconnected" etiketli mesaj kullanıcı tarafından alındığında aşağıda görüldüğü gibi oyunun bittiği belirlenmekte, düğmeler uygun konumlara alınmakta ve yenen tarafın kim olduğunu gösteren mesajlar yazdırılmaktadır (index.js);

```
328   socket.on('Disconneted', function(data){
329     if (GameController.GameOver == BOOL.FALSE){
330       GameController.GameOver = BOOL.TRUE;
331       $('#NewGameButton').attr("disabled", true);
332       $('#Resign').attr("disabled", true);
333       if( playerSide == 0 ){
334         $("#GameStatus").text("OYUN BİTTİ
          {Beyaz yendi - Rakibiniz oyunu terk etti}");
335       }else if( playerSide == 1 ){
336         $("#GameStatus").text("OYUN BİTTİ
          {Siyah yendi - Rakibiniz oyunu terk etti}");
337       }
338     }
339   });
```

4.3 FEN Metninin Girdi Alanı

Kullanıcının bir FEN metnini programa yükleyip kullanabilmesi için bir ara yüze ihtiyaç duyulmaktadır. Bu ara yüz için bir textbox ve button yeterlidir. Bu html nesneleri index.ejs dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
<div id="FenInDiv">
        <input type="text" id="fenIn"/>
        <button type="button" id="SetFen">Set FEN</button>
 </div>
```

SetFen düğmesine basıldığında aşağıdaki jQuery kütüphanesinden faydalanılarak oluşturulmuş olan bir kod çağrılmaktadır. Bu kod olay tabanlı (event triggered) olarak çalışmaktadır. 2. ve 7. satırlar arasındaki kod, bir terfi işlemi için çağrılan menünün oyun tahtası üzerinde oluşturulmuş ihtimaline karşı kullanılmaktadır. Şayet öyle bir menü varsa 2. ve 7. satırlar arasındaki kod bu menüyü ortadan kaldırılmasını sağlamaktadır. 8. satırda fenIn olarak tanımlanan

(id) text alanına girilen değer (val) fenStr değişkenine atanmaktadır. Bu değer boş bir değerse yani, hiçbir değer girilmeden SetFen butonuna basılmışsa kullanıcıya kolaylık olması için standart FEN metni kullanılarak **NewGame(START_FEN)** fonksiyonu çağırılmakta ve oyun başlatılmaktadır (9. ve 10. satırlar). Şayet kullanıcı bir FEN değeri girmişse **NewGame(fenStr)** fonksiyonu o değerle çağrılmaktadır (12. satır). **NewGame** fonksiyonu, eş zamanlı satranç uygulamasının çalıştırılması için diğer fonksiyonların uygun sıra ile çağrıldığı bir fonksiyondur.

```
1  $("#SetFen").click(function () {
2     ClearAllProm();
3     $('.SqSelected').removeClass('SqSelected');
4     moveDone = BOOL.FALSE;
5     PromotionSelectionCalled = BOOL.FALSE;
6     UserMove.from = SQUARES.NO_SQ;
7     UserMove.to = SQUARES.NO_SQ;
8     var fenStr = $("#fenIn").val();
9     if (fenStr == ''){
10          NewGame(START_FEN);
11    }else{
12          NewGame(fenStr);
13    }
14 });
```

Giriş yapılan text alanının ve düğmesinin ekrandaki genişliği, yüksekliği, fontu, rengi, konumu vb. şeylerin ayarlanabilmesi için bir adet css dosyasına ihtiyaç duyulmaktadır. Bu dosya public klasörü içerisine styles.css ismi ile yer almaktadır. Bu dosyanın çağrıldığı index.ejs dosyasındaki head kısmında aşağıdaki link kullanılarak bu dosyaya erişim sağlanmaktadır.

```
<link rel="stylesheet" href="styles.css">
```

styles.css dosyası içerisinde aşağıdaki değerler girilerek index.ejs dosyasındaki FenInDiv isimli div tipindeki kısmın bulunduğu div'in solundan 60 piksel içeriye konuşlanması sağlanmaktadır. Ayrıca fenIn isimli text input nesnesinin genişliği ve sol taraftan ne kadar içeride yer alacağı aşağıdaki şekilde belirlenmektedir.

```
#FenInDiv {
  position: relative;
```

```
  left: 60px;
}

#fenIn {
 width: 480px;
 left: 60px;
}
```

Yukarıdaki düzenleme ile birlikte kullanıcının FEN metnini girebileceği düzgün bir alan oluşturulmaktadır.

4.4 Grafik Ara Yüzlü Oyun Tahtasının Oluşturulması

Öncelikli olarak grafik ara yüzlü oyun tahtasının boyutu ve sınırları belirlenmektedir. Bunun için daha önceden index.ejs dosyası içerisinde oluşturulmuş olan "Board" alanı (div) için styles.css dosyası içerisinde dizayn komutları girilmesi gerekmektedir. "Board" div'i oyun karelerinin içerisine yazdırıldığı alan olduğu için tüm karelerin yerleşebileceği bir alan ayrılması gerekmektedir. Karelerin boyutları 60*60 piksel olarak belirlenmiştir. Oyun tahtasının satır ve sütunları 8*8 olduğu için oyun tahtasının genişlik ve yüksekliğinin 480*480 piksel olması gerekmektedir. Ayrıca oyun tahtasının ekrandaki konumu ve sınır çizgilerinin de belirlenmesi gerekmektedir. Tüm bu işlemeler için gerekli kodlar aşağıda görüldüğü şekilde styles.css dosyası içerisinde tanımlanmaktadır;

```
#Board {
    position: relative;
    top: 5px;
    left: 60px;
    width: 480px;
    height: 480px;
    border-style: solid;
    border-width: 2px;
}
```

Grafik ara yüzlü oyun tahtasını oluşturan kareler bir fonksiyon kullanılarak oluşturulmaktadır. Bunun için main.js dosyası içerisindeki InitBoardSquares fonksiyonu kullanılmaktadır;

```
1  Function InitBoardSquares() {
2      var light = 1;
3      var rankName;
4      var fileName;
5      var divString;
6      var rankIter;
7      var fileIter;
8      var lightString;
9
10     for(rankIter = RANKS.RANK_8;rankIter >= RANKS.RANK_1;rankIter--){
11         light ^= 1;
12         rankName = "rank" + (rankIter + 1);
13         for(fileIter = FILES.FILE_A; fileIter <= FILES.FILE_H; fileIter++) {
14             fileName = "file" + (fileIter + 1);
15             if(light == 0) lightString="Light";
16             else lightString = "Dark";
17             light^=1;
18             divString = "<div class=\"Square " + rankName + " " +
               fileName + " " + lightString + "\"/>";
19             $("#Board").append(divString);
20         }
21     }
22 }
```

InitBoardSquares fonksiyonunda html formatına uygun olarak üretilen metinler sayesinde oyun tahtası oluşturulmaktadır. Hazırlanan bu html ifadede kullanılan kelimelerden biri olan "rank(1-8)" oyun tahtasındaki her bir satırı temsil etmektedir ve her bir rank değerinin grafik kullanıcı ara yüzünde bulunacağı pozisyon styles.css dosyasında aşağıdaki şekilde belirlenmektedir;

```
.rank1 { top: 420px; }
.rank2 { top: 360px; }
.rank3 { top: 300px; }
.rank4 { top: 240px; }
.rank5 { top: 180px; }
.rank6 { top: 120px; }
.rank7 { top: 60px; }
.rank8 { top: 0px; }
```

Yukarıda rank1'den rank8'e kadar olan satırların oyun tahtası üzerinde yukarıdan aşağıya doğru kaçıncı pikselde bulunacağı belirlenmektedir. Buna göre rank8 en üste, rank1 ise en alta yerleştirilmektedir. rank'lerin arasındaki uzaklığa

bakıldığında her bir karenin 60 piksel kenar ölçüsü olduğu anlaşılmaktadır. Burada belirtilen "top" (yukarıdan başlama noktası), "rank(1-8)" ismiyle sınıflandırılan html nesnesinin bulunduğu alan (div) içerisinde yukarıdan aşağıya doğru nerede bulunacağını belirtmektedir. Benzer şekilde oyun tahtası üzerindeki satırlar da tanımlanmaktadır. Satırları tanımlamak için "file(1-8)" kullanılmakta ve styles.css dosyası içerisinde aşağıdaki şekilde oluşturulmaktadır;

```
.file1 { left: 0px; }
.file2 { left: 60px; }
.file3 { left: 120px; }
.file4 { left: 180px; }
.file5 { left: 240px; }
.file6 { left: 300px; }
.file7 { left: 360px; }
.file8 { left: 420px; }
```

Yukarıdaki bu tanımlama ile her bir "file" sınıfının içinde bulunduğu alan (div) içerisinde o div'in soldan hangi konumda olacağı belirlenmektedir. rank ve file sınıfları kesiştirildiğinde bir oyun karesinin nerede konumlanacağı ortaya çıkmaktadır. Ayrıca **InitBoardSquares** fonksiyonu içerisinde "Dark" ve "Light" ifadeleri de kullanılmaktadır. Bu ifadeler styles.css dosyası içerisinde aşağıdaki şekilde tanımlanmaktadır;

```
.Dark {
    background-color:#5fbcd3;
}

.Light {
    background-color:#afdde9;
}
```

Fonksiyon sayesinde bir kareye "Dark" sınıfı eklenerek koyu renk kareler oluşturulmakta, Ligth sınıfı eklenerek de açık renkteki kareler oluşturulmaktadır.

InitBoardSquares fonksiyonu içerisinde geçen bir diğer ifade de " Square" ifadesidir ve styles.css dosyası içerisinde aşağıdaki şekilde dizayn özellikleri tanımlanmaktadır;

```
.Square {
   position: absolute;
```

```
    width: 60px;
    height: 60px;
}
```

Square sınıfı "absolute" pozisyon özelliğine sahiptir. Bunun anlamı; html etiketine kadar olan hiç bir üst konteynır relative pozisyon özelliğine sahip değilse ekranda tanımlandığı yere yerleşir ve tüm satırı kaplamak yerine sadece kendine tanımlı alanı kaplar. Burada bir üst konteynır olan "Board" relative pozisyon özelliğine sahip olduğu için Square sınıfı ile tanımlı olan her alan "Board" div'i içerisinde bir pozisyon almaktadır ve sadece 60px*60px bir alan kaplamaktadır.

InitBoardSquares fonksiyonunun 2. ve 8. satırları arasında fonksiyon içerisinde kullanılan değişkenler tanımlanmaktadır. Oyun tahtasındaki tüm satır ve sütunların gezilebilmesi için iç içe iki for döngüsü tanımlanmaktadır. Dıştaki for döngüsü ile tüm satırlar, içteki for döngüsü ile tüm sütunlar gezilmektedir. 10. satırdaki dış for döngüsü ile oyun tahtasının 8. satırından başlanılarak 1. satırına kadar gidilmektedir. 11. satırda "light" değişkeni "0" ise "1", "1" ise "0" yapılmaktadır. "0" değeri oluşturulan karenin açık renkte olacağını "1" değeri ise koyu renkte olacağını temsil etmektedir. Bu kod satırı, oyun tahtasında her satır başlangıcında oluşturulacak olan karenin renginin bir üst satırdan farklı olmasını sağlamak için kullanılmaktadır. 12. satırda, rankName değişkenine hangi satıra ait oyun karesi oluşturulacaksa o satırın ismi atanmaktadır. Bu isimler "rank1 , rank2, rank3, rank4, rank5, rank6, rank7, rank8" şeklindedir. Bu satırlar "class" isimlerini temsil etmektedir ve bu class'lar oluşturulan oyun karesinin yukarıdan kaç piksel aşağıda olacağını belirtmektedir. 13. satırda iç for döngüsü başlatılmaktadır. Bu for döngüsüyle "A" sütunundan başlanarak "H" sütunu dahil tüm sütunlar gezilmektedir. 14. satırda, fileName değişkenine hangi sütuna ait oyun karesi oluşturulacaksa o sütunun ismi atanmaktadır. Bu isimler "file1, file2, file3, file4, file5, file6, file7, file8" şeklindedir. 15. ve 16. satırlarda, "light" değişkeninin değerine göre "lightString" değişkenine atanacak olan metin belirlenmektedir. 17. satırda, aynı oyun tahtası satırında bir sonraki karenin rengini açık renkse koyu, koyu renkse açık renk yapmak için "light" değişkeninin değeri değiştirilmektedir. 18. satırda, fonksiyonun bu aşamasına kadar elde edilen değerler ve "Square" kelimesi, html formatına uygun bir şekilde yan yana getirilerek "divString" değişkenine atanmaktadır. 19. satırda da divString değişkeni "Board" div'ine eklenmektedir. Böylelikle oyun tahtası oluşturulmaktadır. İlk dış for döngüsü ve 8 iç for döngüsü çalışması sonucunda oluşturulan metnin html formatındaki görünümü aşağıdaki şekilde olmaktadır;

```
<div id="Board">
  <div class="Square rank8 file1 Light"></div>
  <div class="Square rank8 file2 Dark"></div>
  <div class="Square rank8 file3 Light"></div>
  <div class="Square rank8 file4 Dark"></div>
  <div class="Square rank8 file5 Light"></div>
  <div class="Square rank8 file6 Dark"></div>
  <div class="Square rank8 file7 Light"></div>
  <div class="Square rank8 file8 Dark"></div>
.
.
.
</div>
```

Şu ana kadar oyun tahtası ve içerisindeki oyun kareleri oluşturulmuş bulunmaktadır. Grafik kullanıcı ara yüzünde kullanılan oyun taşlarının oyun tahtasında gösterimi ise her bir oyun taşı için kullanılan "png" formatındaki resim dosyaları ile yapılmaktadır. Bu resim dosyalarının ismi kodlama esnasında kullanılan oyun taşı isimleriyle aynıdır. Örneğin: "wP.png" beyaz piyon resminin kayıtlı olduğu dosyadır. Grafik kullanıcı ara yüzünde, oyun taşlarının oyun tahtası üzerine yerleştirilmesi aşağıda gösterilen, gui.js dosyası içerisinde kodlanış olan, **SetInitialBoardPieces** fonksiyonu ile yapılmaktadır;

```
1  function SetInitialBoardPieces() {
2
3      var sq;
4      var sq120;
5      var file,rank;
6      var rankName;
7      var fileName;
8      var imageString;
9      var pieceFileName;
10     var pce;
11
12     ClearAllPieces();
13
14     for(sq = 0; sq < 64; ++sq) {
15         sq120 = SQ120(sq);
16         pce = GameBoard.pieces[sq120];
17
```

```
18          if(GameController.BoardFlipped == BOOL.TRUE) {
19              sq120 = MIRROR120(sq120);
20          }
21
22          if(pce >= PIECES.wP && pce <= PIECES.bK) {
23              AddGUIPiece(sq120, pce);
24          }
25      }
26  }
```

SetInitialBoardPieces fonksiyonunun 3. ve 10. satırları arasında fonksiyonda kullanılan değişkenler tanımlanmaktadır. 12. satırda, ClearAllPieces fonksiyonu çağrılarak oyun tahtası üzerindeki tüm oyun taşları temizlenmektedir. Bu işlem, daha önceden oyun tahtasında olan taşların üzerine yeni taşlar eklememek için yapılmaktadır. **ClearAllPieces** fonksiyonu aşağıda görüldüğü gibi gui.js dosyası içerisinde yer almaktadır;

```
function ClearAllPieces() {
    $(".Piece").remove();
}
```

ClearAllPieces fonksiyonunda jQuery kütüphanesinden remove fonksiyonu kullanılmaktadır. Bu fonksiyonla "Piece" class'ına sahip tüm alanlar html alanından silinmektedir. 14. satırda, oyun tahtasındaki aktif olarak kullanılan tüm oyun karelerinin gezilmesi için bir for döngüsü başlatılmaktadır. 15. satırda indis numaraları 120 elemanlı oyun tahtası dizisine göre değiştirilmektedir. 64 elemanlı dizideki "0", 120 elemanlı dizideki "21"e karşılık gelmektedir. 16. satırda, oyun tahtasındaki ilk karedeki bilgi "pce" değişkenine atanmaktadır. Bu bilgi bir boş kare bilgisi veya bir oyun taşı bilgisi olabilmektedir. Şayet oyun tahtası ters çevrilmişse sq120 değişkenindeki oyun karesi bilgisi de ter çevrilmektedir (18. ve 20. satırlar arası). Bu işlem main.js içerisindeki **MIRROR120** fonksiyonu ile yapılmaktadır. Kullanılan dizinin son eleman numarası olan 119'dan oyun karesi numarası çıkarılarak tahta ters çevrildiğinde o karenin nereye denk geldiği bulunmaktadır. Fonksiyon aşağıdaki gibidir;

```
function MIRROR120(sq) {
    return 119 - sq;
}
```

Şayet "pce" değişkenine atanan değer bir oyun taşı değeri ise **AddGUIPiece** fonksiyonu kullanılarak o taş grafik ara yüzde "sq120" değişkeni ile gösterilen yere yerleştirilmektedir. **AddGUIPiece** fonksiyonu gui.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function AddGUIPiece(sq, pce) {
2
3      var file = FilesBrd[sq];
4      var rank = RanksBrd[sq];
5      var rankName = "rank" + (rank+1);
6      var fileName = "file" + (file+1);
7      var pieceFileName = "images/" + SideChar[PieceCol[pce]] +
       PceChar[pce].toUpperCase() + ".png";
8      var imageString = "<image src=\"" + pieceFileName + "\" class=\"Piece " +
       rankName + " " + fileName + "\"/>";
9      $("#Board").append(imageString);
10 }
```

AddGUIPiece fonksiyonunun 3. satırında FilesBrd dizisi yardımıyla oyun taşının bulunduğu karenin hangi oyun tahtası sütununda olduğu "file" değişkenine atanmaktadır. 4. satırda, benzer şekilde bu sefer oyun taşının hangi oyun tahtası satırında olduğu "rank" değişkenine atanmaktadır. 5. ve 6. satırlarda, oyun tahtası satırlarına ve sütunlarına ait isimler, styles.css dosyası içerisindeki class tanımına uygun hale getirilmektedir. 7. satırda, oyun taşına ait resim dosyasının yolu ve ismi pieceFileName değişkenine atanmaktadır. 8. satırda, oyun taşının dosya sistemi içerisindeki yolunun ve isminin kayıtlı olduğu "pieceFileName" değişkeni ile "Piece", "rankName" ve "fileName" değişkenlerinde tutulan üç adet class html formatına uygun şekilde "imageString" değişkenine atanmaktadır. 9. satırda ise "imageString" değişkeni "Board" div alanına eklenmektedir. "Board" div alanına eklenen bu metinler html sayfasında aşağıdaki şekilde görülmektedir;

```
<div id="Board">
  <img src="images/wR.png" class="Piece rank1 file1">
  <img src="images/wN.png" class="Piece rank1 file2">
  <img src="images/wB.png" class="Piece rank1 file3">
  <img src="images/wQ.png" class="Piece rank1 file4">
  <img src="images/wK.png" class="Piece rank1 file5">
  <img src="images/wB.png" class="Piece rank1 file6">
  <img src="images/wN.png" class="Piece rank1 file7">
  <img src="images/wR.png" class="Piece rank1 file8">
```

```
.
.
.
</div>
```

4.5 Grafik Kullanıcı Ara Yüzünde Hamlelerin Yapılması

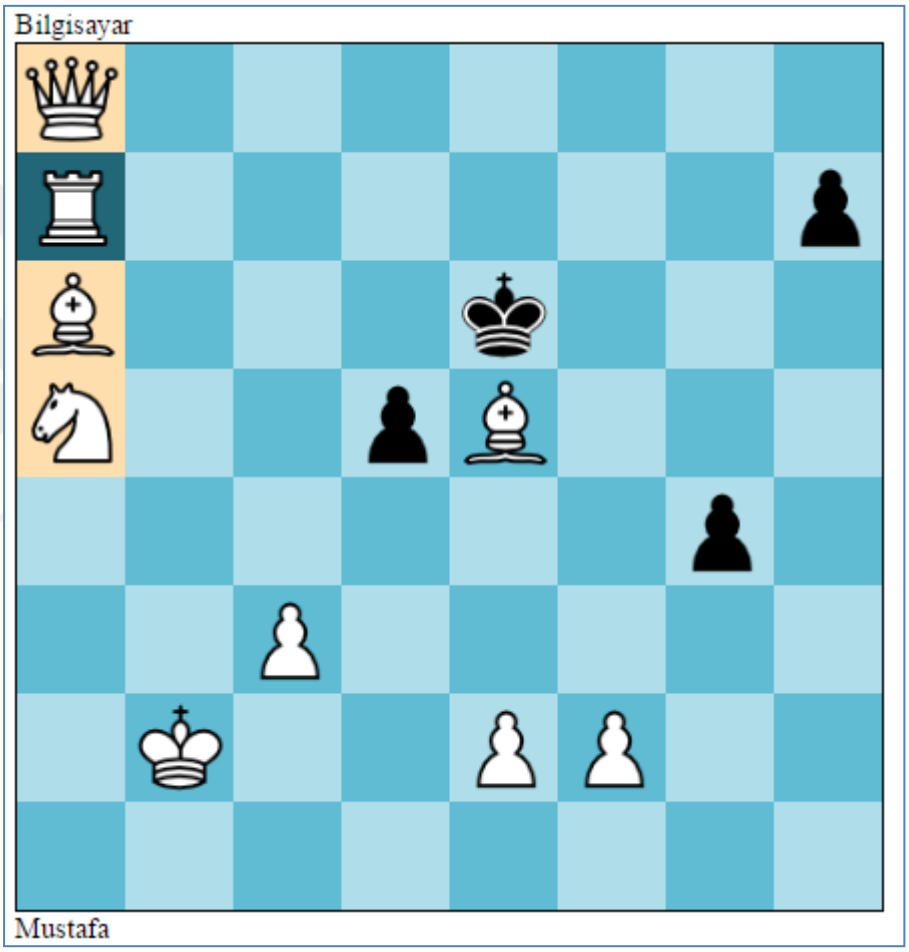
Grafik kullanıcı ara yüzünde bir hamle yapılabilmesi için önce bir oyun taşı tıklanmalı ardından da gidebileceği bir boş kareye veya yiyebileceği bir rakip taşın üzerine tıklanması gerekmektedir. Bir taşın üzerinde tıklanma işleminde seçici olarak "Piece" class'ı, bir oyun karesi üzerine tıklanma işleminde ise seçici olarak "Square" class'ı kullanılmaktadır. Aşağıda bir "Piece" class'ına tıklandığında hangi işlemlerin yapıldığı görülmektedir (gui.js);

```
1  $(document).on('click','.Piece', function (e) {
2      if ( moveDone == BOOL.FALSE  &
       PromotionSelectionCalled == BOOL.FALSE ){
3          console.log('Piece Click');
4          console.log("e.pageX: " + e.pageX +" "+ "e.pageY : "+ e.pageY);
5
6          if(UserMove.from == SQUARES.NO_SQ) {
7              UserMove.from = ClickedSquare(e.pageX, e.pageY);
8          } else {
9              UserMove.to = ClickedSquare(e.pageX, e.pageY);
10         }
11
12         if(UserMove.from != SQUARES.NO_SQ &&
           UserMove.to != SQUARES.NO_SQ){
13             uPreMv = ParseMove(UserMove.from,UserMove.to);
14             if(uPreMv != NOMOVE){
15                 moveDone = BOOL.TRUE;
16                 if(PROMOTED(uPreMv) != PIECES.EMPTY){
17                     PromotionSelection(UserMove.from,UserMove.to);
18                 }else {
19                     uMv = uPreMv;
20                     checkOppMoveAndPlay();
21                 }
22             }else{
23                 DeselectSq(UserMove.from);
24                 DeselectSq(UserMove.to);
25                 UserMove.from = SQUARES.NO_SQ;
26                 UserMove.to = SQUARES.NO_SQ;
27             }
```

```
28
29              }
30          }
31  });
```

Birinci satırda, bir "Piece" class özelliğine sahip bir alana tıklandığında anonim bir fonksiyon başlatılmaktadır. Şayet moveDone değişkeni BOOL.FALSE değerine sahipse yani, mevcut oyun pozisyonunda yapılmak istenilen hamle henüz tamamlanmamışsa, aynı zamanda PromotionSelectionCalled değişkeni de BOOL.FALSE değerine sahipse yani, bir terfi işlemi için terfi edilecek taşın seçildiği menü grafik kullanıcı ara yüzde görünmüyorsa (2. satır) tıklanılan taş üzerinde işlem yapmaya başlanmaktadır. moveDone değişkeni ile yapılan kontrol, oyun esnasında bir hamle yapıldıktan sonra oyuncunun fikrini değiştirip başka bir hamle yapmasını engellemek için yapılmaktadır. İzin verildiği durumlarda, özellikle ikili oyunlarda, karşı tarafa yapılan ilk hamle gönderilirken oyuncunun kendi tahtası üzerinde fikrini değiştirdikten sonra yaptığı ikinci hamle işleme alınmaktadır. Bu yanlış durum moveDone değişkeni yardımıyla düzeltilmektedir. PromotionSelectionCalled yapılan kontrol ise ekranda bir terfi menüsü açıkken kullanıcıyı o terfi menüsünden seçim yapmaya zorlamak için kullanılmaktadır. Aksi takdirde, seçim menüsü açıkken kullanıcı aynı zamanda bir hamle yaptığında grafik ara yüzde istenmeyen bir görüntü ortaya çıkmaktadır. 3. satırda, bir "Piece" alanının tıklandığı konsola yazdırılmaktadır. 4. satırda, tıklanılan alanın web sayfası üzerindeki X ve Y koordinatları konsola yazdırılmaktadır. Bu iki satır programcıya bilgi vermek amaçlıdır. 6. satırda, UserMove.from değişkeninin boş mu dolu mu olduğu kontrol edilmektedir. UserMove.from değişkeni bir taşın nereden harekete başladığını tutmak için kullanılmaktadır. Şayet UserMove.from değişkeni boşsa (6. satır) bunun anlamı, üzerine tıklanılmış olan kare hamlenin başlangıç noktasıdır ve bu kare UserMove.from değişkenine atanmaktadır (7. satır). Şayet UserMove.from değişkeni boş değilse (8. satır), tıklanılmış olan "Piece" class özelliğine sahip olan taş rakibin taşıdır ve hamlenin yapıldığı kareyi göstermektedir. Bu nedenle tıklanılan kare bilgisi hamlenin yapıldığı karenin tutulduğu UserMove.to değişkenine atanmaktadır (9. satır). Tıklanılan karenin hangi kare olduğunun bulunması için **ClickedSquare** fonksiyonu kullanılmaktadır. Şayet hem UserMove.from hem de UserMove.to değişkenlerinde bir oyun karesi değeri varsa (12. satır) yani, nereden nereye hamle yapılacağı belirlenmişse, 13. ve 21. satırlar arasındaki kod çalıştırılmaktadır. 13. satırda, yapılmak istenilen hamlenin geçerli bir hamle olup olmadığı **ParseMove** fonksiyonu yardımı ile kontrol edilmekte ve sonuç uPreMv değişkenine

atanmaktadır. Şayet uPreMv NOMOVE değerinden farklı bir değere sahipse yani, yapılmak istenilen hamle geçerli bir hamle ise (14. satır) 15. ve 21. satırlar arasındaki kod çalıştırılmaktadır. 15. satırda, geçerli hamle yapıldıktan sonra aynı pozisyon için başka bir hamle yapılmasının önüne geçmek için kullanılan moveDone değişkenine BOOL.TRUE değeri atanmaktadır. Şayet yapılan hamle sonucu bir terfi işlemi söz konusu ise (16. satır) **PromotionSelection** fonksiyonu çağrılarak terfi edilebilecek taşların listesi Şekil 4-6'de görüldüğü gibi oyun tahtası üzerinde sıralanmaktadır (17. satır).



Şekil 4-6 Terfi işlemi menüsü

Kullanıcı tarafından, bu menüdeki taşlardan birisi seçilerek terfi edilen taş belirlenmektedir (PieceProm class'ına ait bir alana tıklama işlemi). Şayet bir terfi işlemi yoksa (18. satır) uPreMv değişkenindeki değer uMv değişkenine atanarak kullanıcı hamlesinin son durumu belirlenmiş olmaktadır (19. satır). 20. satırda, **checkOppMoveAndPlay** fonksiyonu çağrılmaktadır. **checkOppMoveAndPlay** fonksiyonu ile karşı tarafın hamlesini yapıp yapmadığı kontrol edilmektedir. Şayet

karşı taraf hamlesini yapmışsa hamleler oyun tahtası üzerinde işlenmektedir. Şayet oyunda terfi işlemi varsa bu işlemler PieceProm class'ına sahip bir alana tıklandığında da yapılmaktadır. Şayet oyuncu tarafından geçerli bir hamle yapılamamışsa (22. satır) tahta üzerinde seçilmiş olan karelerin seçimi kaldırılmakta (23. ve 24. satırlar) ve hamlenin başlangıç ve bitiş karelerini gösteren değişkenler sıfırlanmaktadır (25. ve 26. satırlar).

Piece class'ına sahip bir alana tıklandığında çalıştırılan anonim fonksiyon tarafından çağrılan **ClickedSquare** fonksiyonu gui.js dosyası içerisinde aşağıdaki gibi yer almaktadır. ClickedSquare fonksiyonu, html sayfası üzerinde tıklanılan yerin, oyun tahtası üzerinde hangi kare olduğunun bulunması için kullanılmaktadır;

```
1  function ClickedSquare(pageX, pageY) {
2      console.log('ClickedSquare() at ' + pageX + ',' + pageY);
3      var position = $('#Board').position();
4
5      var workedX = Math.floor(position.left);
6      var workedY = Math.floor(position.top);
7
8      pageX = Math.floor(pageX);
9      pageY = Math.floor(pageY);
10     console.log('PageX pageY after floor ' + pageX + ',' + pageY);
11     var file = Math.floor((pageX-workedX) / 60);
12     console.log("Math.floor((pageX-workedX) / 60)" + file);
13     var rank = 7 - Math.floor((pageY-workedY) / 60);
14     console.log("7 - Math.floor((pageY-workedY) / 60)" + rank);
15     var sq = FR2SQ(file,rank);
16
17     if(GameController.BoardFlipped == BOOL.TRUE) {
18         sq = MIRROR120(sq);
19     }
20
21     console.log('Clicked sq:' + PrSq(sq));
22
23     if (GameController.GameOver == BOOL.FALSE){
24         console.log('return sq' + sq);
25         SetSqSelected(sq);
26         return sq;
27     }
28 }
```

ClickedSquare fonksiyonu, html sayfası üzerinde tıklanılmış olan X ve Y koordinatları parametre olarak verilerek başlatılmaktadır. X koordinatında sayfanın en solu "0" olarak alınmakta ve sağa doğru ilerlemektedir. Y koordinatında ise sayfanın en üstü "0" olarak alınmakta ve aşağı doğru ilerlenmektedir. 3. satırda, "Board" div alanının html sayfası üzerinde nerede bulunduğuna ait pozisyon bilgisi "position" değişkenine atanmaktadır. 5. ve 6. satırlarda, bu pozisyon bilgisinin X ve Y koordinatları workedX ve workedY değişkenlerine atanmaktadır. Bu atamalarda **Math.floor** fonksiyonu kullanılarak sayılar aşağı yuvarlanmaktadır. 8. ve 9. satırlarda, tıklanılan alanlara ait koordinat bilgileri pageX ve pageY değişkenlerine aşağı yuvarlanarak atanmaktadır. 10, 12, 14, 21 ve 24. satırlar konsola yazdırılan bilgi satırlarıdır. 11. satırda, tıklanılan alanın oyun tahtası üzerindeki sütun değeri file değişkenine atanmaktadır. Bunun için pageX değişkenindeki koordinat değeri workedX değerinden çıkartılarak oyun tahtası üzerinde X ekseni üzerinde nereye tıklandığı bulunmaktadır. Bu değer 60'a bölünerek tıklanılan alanın hangi sütuna denk geldiği hesaplanmaktadır. Board div alanının içerisinde kareler X=0px ve Y=0px alınarak yerleştirilmeye başlandığı ve her kare 60px'lik bir alan kapladığı için bu işlem bize doğru sütunu vermektedir. 13. satırda, tıklanılan alanın oyun tahtası üzerindeki satır değeri rank değişkenine atanmaktadır. Burada sütun değerinin hesaplanmasına benzer bir hesaplama yapılmaktadır ancak bulunan değer 7'den çıkarılarak satır değerine ulaşılmaktadır. Bunun nedeni ise satırların ekrana 8. satırdan başlanarak yazdırılmasıdır. Örneğin; sol üst köşedeki kareye tıklandığında pageY'nin pageX'den çıkarılması sonucunda "0" değerine ulaşılacaktır. "0/60" da "0" olacağından dolayı tıklanılan alanın birinci ("0" rakamı ile temsil edilmektedir) satır olduğu düşünülecektir. Ama grafik kullanıcı ara yüzünde yukardan aşağıya doğru 8.satırdan (7 rakamı ile temsil edilmektedir.) başlanılarak satırlar yerleştirildiği için o alan 8. oyun tahtası satırını temsil etmektedir. Dolayısıyla bulunan değer 7'den çıkarıldığında doğru oyun tahtası satırı bulunmuş olmaktadır. Örneğimizde "7 -0 = 7" olduğu için o alanın 8. oyun tahtası satırı olduğu anlaşılmaktadır. 15. satırda, elde edilen satır ve sütun değerleri kullanılarak tıklanılan oyun karesi bilgisi elde edilmekte ve "sq" değişkenine atanmaktadır. Şayet oyun tahtası ters çevrilmişse (17. satır) tıklanılmış olan kare değeri "sq" değişkenine **MIRROR120** fonksiyonu yardımıyla atanmaktadır. Şayet oyun devam ediyorsa (23. satır) tıklanılmış olan kare **SetSqSelected** fonksiyonu ile seçilmekte (koyu bir renk ile boyanmakta) (25. satır) ve "sq" bilgisi döndürülmektedir. **SetSqSelected** fonksiyonu gui.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function SetSqSelected(sq) {
2
3    if(GameController.BoardFlipped == BOOL.TRUE) {
4        sq = MIRROR120(sq);
5    }
6
7    $( ".Square" ).each(function( index ) {
8        if( (RanksBrd[sq] == 7 - Math.round($(this).position().top/60)) &&
         (FilesBrd[sq] == Math.round($(this).position().left/60)) ){
9            $(this).addClass('SqSelected');
10     }
11    });
12 }
```

Seçilmiş olan bir karenin seçimini kaldırmak için de gui.js dosyası içerisinde aşağıdaki şekilde yer alan **DeselectSq** fonksiyonu kullanılmaktadır;

```
1  function DeselectSq(sq) {
2
3    if(GameController.BoardFlipped == BOOL.TRUE) {
4        sq = MIRROR120(sq);
5    }
6
7    $( ".Square" ).each(function( index ) {
8     if( (RanksBrd[sq] == 7 - Math.round($(this).position().top/60)) &&
      (FilesBrd[sq] == Math.round($(this).position().left/60)) ){
9            $(this).removeClass('SqSelected');
10     }
11    });
12 }
```

SetSqSelected fonksiyonu ile tıklanılan kareye SqSelected class'ı eklenmektedir. **DeselectSq** fonksiyonu ile de SqSelected class'ı silinmektedir. SqSelected class'ı eklenildiği karenin rengini biraz daha koyu yapmaktadır. SqSelected class'ı styles.css dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
.SqSelected {
    background-color:#216778;
}
```

Oyuncu tarafından yapılan bir hamlenin geçerli bir hamle olup olmadığı kontrolünü yapan **ParseMove** fonksiyonu io.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
 1  function ParseMove(from, to) {
 2
 3      GenerateMoves();
 4
 5      var Move = NOMOVE;
 6      var PromPce = PIECES.EMPTY;
 7      var found = BOOL.FALSE;
 8
 9      for(index = GameBoard.moveListStart[GameBoard.ply];
        index < GameBoard.moveListStart[GameBoard.ply + 1]; ++index) {
10          Move = GameBoard.moveList[index];
11          if(FROMSQ(Move) == from && TOSQ(Move) == to) {
12              PromPce = PROMOTED(Move);
13              if(PromPce != PIECES.EMPTY) {
14                  if( (PromPce == PIECES.wQ &&
                    GameBoard.side == COLOURS.WHITE) ||
                    (PromPce == PIECES.bQ &&
                    GameBoard.side == COLOURS.BLACK) ) {
15                      found = BOOL.TRUE;
16                      break;
17                  }
18                  continue;
19              }
20              found = BOOL.TRUE;
21              break;
22          }
23      }
24
25      if(found != BOOL.FALSE) {
26          if(MakeMove(Move) == BOOL.FALSE) {
27              return NOMOVE;
28          }
29          TakeMove();
30          return Move;
31      }
32
33      return NOMOVE;
34  }
```

ParseMove fonksiyonunun 3. satırında, hamleyi yapan tarafın geçerli tüm hamlelerini üretmek için **GenerateMoves** fonksiyonu çağrılmaktadır. 9. ve 23. satırlar arasında, yapılan hamle bulunana kadar, mevcut oyun tahtası pozisyonunda, geçerli tüm hamleler kontrol edilmekte ve yapılan hamle yapılabilecek hamleler listesinde bulunursa döngüden çıkılmaktadır. Bu döngü içerisinde 10. ve 19. satırlar arasında, yapılan hamlede bir terfi işlemi varsa terfi edilen taş olarak vezir seçilmektedir. Bu işlem sadece her ne kadar her iki taraf için uygulansa da sadece bilgisayar tarafı için aynı şekilde bırakılarak işleme alınmaktadır. Çünkü oyuncu için seçim yapabileceği bir menü çıkarılarak seçim yaptırılmakta ve o seçim sonucu işleme alınmaktadır. Şayet yapılan hamle yapılabilecek hamleler listede bulunmuşsa (25. satır) **MakeMove** fonksiyonu kullanılarak hamle klasik satranç oyunu kuralları çerçevesinde yapılmaktadır. Eş zamanlı satranç oyunu ile klasik satranç oyunundaki mat etme kuralları örtüştüğü için bu şekilde bir kontrol uygun düşmektedir. Eğer hamle geçerliyse yani, hamle sonucu oyuncu mat olmuyorsa **TakeMove** fonksiyonu ile hamle geri alınmakta (29. satır) ve hamle döndürülmektedir. Şayet hamle geçersizse (26. satır) NOMOVE değeri döndürülmektedir (27. satır). Fonksiyonun yukarısında hiçbir dönüş yapılmamışsa 33. satırda NOMOVE değeri döndürülmektedir. Fonksiyonun buraya kadar ulaşması olası değildir ama tedbir amacıyla konulmuş bir satırdır.

PromotionSelection fonksiyonu gui.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
 1  function PromotionSelection(from,to){
 2      PromotionSelectionCalled = BOOL.TRUE;
 3      var tempTo = to;
 4      if( PieceCol[GameBoard.pieces[from]] == COLOURS.WHITE){
 5          for(var i = 0 ; i<4 ; i++){
 6              RemoveGUIPiece(tempTo);
 7              SetSqProm(tempTo);
 8              AddGUIPieceProm(tempTo, PcePromW[i]);
 9              tempTo -= 10;
10          }
11      }
12      if( PieceCol[GameBoard.pieces[from]] == COLOURS.BLACK){
13          for(var i = 0 ; i<4 ; i++){
14              RemoveGUIPiece(MIRROR120(tempTo));
15              SetSqProm(tempTo);
16              AddGUIPieceProm(MIRROR120(tempTo), PcePromB[i]);
```

```
17              tempTo += 10;
18          }
19      }
20  }
```

PromotionSelection fonksiyonunun ikinci satırında, terfi menüsünün çağrıldığını belirten değişken BOOL.TRUE yapılmaktadır. Bu değişken menüden bir seçim yapılmadan başka bir işlem yapılmasını önlemek için kullanılmaktadır. Fonksiyon içerisinde beyaz taraf (4. ve 10. satırlar arası) ve siyah taraf için (12. ve 19. satırlar) aynı işlemleri yapan iki adet for döngüsü kullanılmaktadır. 6. satırda, menünün oluşturulacağı kareler üzerindeki tüm taşlar grafik ara yüzden silinmektedir. Bunun amacı taşların üst üste gelip kötü bir görüntü oluşturmasını önlemektir. 7. satırda, işlem yapılan kareye **SetSqProm** fonksiyonu yardımıyla SqProm class'ı eklenmektedir. 8. satırda, menüde yer alacak olan taşlar menüye eklenmektedir. Fonksiyonun geri kalanında aynı işlemler siyah taraf için yapılmaktadır. Aşağıdaki gösterilen, gui.js dosyası içerinde yer alan **SetSqProm** fonksiyonunda işlem yapılan kareye ait grafik kullanıcı ara yüzündeki "Square" class'ına sahip div bulunarak o div'e SqProm class'ı eklenmektedir.

```
1  function SetSqProm(sq) {
2      if(GameController.BoardFlipped == BOOL.TRUE) {
3          sq = MIRROR120(sq);
4      }
5
6      $( ".Square" ).each(function( index ) {
           if( (RanksBrd[sq] == 7 - Math.round($(this).position().top/60)) &&
7          (FilesBrd[sq] == Math.round($(this).position().left/60)) ){
8              $(this).addClass('SqProm');
9          }
10     });
11 }
```

SqProm class'ı kullanılarak terfi taşlarının yer alacağı menünün zemin rengi değiştirilmektedir. Aynı zamanda hangi taşın seçildiği de SqProm class'ı yardımıyla bulunmaktadır. SqProm class'ı styles.css dosyası içerinde aşağıdaki şekilde yer almaktadır;

```
.SqProm{
    background-color:#FFDEAD;
}
```

Terfi edilecek taşların menü üzerine yerleştirilmesinde kullanılan **AddGUIPieceProm** fonksiyonu gui.js dosyası içerinde aşağıdaki şekilde yer almaktadır;

```
1  function AddGUIPieceProm(sq, pce) {
2      var file = FilesBrd[sq];
3      var rank = RanksBrd[sq];
4      var rankName = "rank" + (rank+1);
5      var fileName = "file" + (file+1);
       var pieceFileName = "images/" + SideChar[PieceCol[pce]] +
6      PceChar[pce].toUpperCase() + ".png";
       var imageString = "<image src=\"" + pieceFileName + "\" class=\"
7      PieceProm PiecePromHover " + rankName + " " + fileName + "\"/>";
8      $("#Board").append(imageString);
9  }
```

AddGUIPieceProm fonksiyonu ile taşlar, terfi menüsüne "PieceProm" ve "PiecePromHover" class'larına sahip imaj alanları içerisinde eklenmektedir. Taşlar üzerinde gezildiğinde ve taşlara tıklandığında bu class'lardan faydalanılarak işlem yapılmaktadır. Bu iki class aşağıdaki şekilde styles.css dosyası içerisinde yer almaktadır.

```
.Piece, .PieceProm {
    position: absolute;
}
.PiecePromHover:hover{
    cursor: pointer;
    background-color:#216778;
}
```

Oluşturulan bu menüden terfi edilecek taş üzerine tıklandığında yani, .PieceProm class'ına sahip bir alan tıklandığında aşağıdaki işlemler yapılmaktadır (gui.js);

```
1  $(document).on('click','.PieceProm', function (e) {
2      $(".PiecePromHover").removeClass("PiecePromHover");
3      if(ProSelectDone == BOOL.FALSE){
4          ProSelectDone = BOOL.TRUE;
5          tempTo = UserMove.to;
6          clkSq = ClickedSquare(e.pageX, e.pageY);
7          SetSqPromSelected(clkSq);
```

```
 8          if( PieceCol[GameBoard.pieces[UserMove.from]] ==
            COLOURS.WHITE){
 9            for(var i = 0 ; i<4 ; i++){
10                if(clkSq == tempTo ){
11                    uMv = PROMOTION(uPreMv, PcePromW[i]);
12                    if(player == 1 || (player == 2 & oppMove != NOMOVE)){
13                        ClearAllProm();
14                    }
15                    break;
16                }
17                tempTo -= 10;
18            }
19          }else if( PieceCol[GameBoard.pieces[UserMove.from]] ==
            COLOURS.BLACK){
20             for(var i = 0 ; i<4 ; i++){
21                if(clkSq == tempTo ){
22                   uMv = PROMOTION(uPreMv, PcePromB[i]);
23                   if(player == 1 || (player == 2 & oppMove != NOMOVE)){
24                      ClearAllProm();
25                   }
26                   break;
27                }
28                tempTo += 10;
29             }
30          }
31
32          checkOppMoveAndPlay();
33       }
34  });
```

Yukarıda PieceProm class'ına sahip bir alana tıklanılarak başlatılan anonim fonksiyonun 2. satırında, artık terfi edilecek olan taş seçimi yapıldığı için taşların üzerinde gezinirken o alanı koyulaştırmaya yarayan PiecePromHover class'larına ihtiyaç kalmadığından dolayı silinmektedir. Şayet terfi seçimi yapılıp yapılmadığını gösteren değişken seçim yapılmadığını gösteriyorsa (3. satır) 4. satırdan itibaren devam edilmektedir. 4. satırda, terfi için bir seçim yapıldığı işaretlenmektedir. 5. satırda, hamlenin yapıldığı kare tempTo değişkenine atanmaktadır. 6. satırda, hangi oyun karesine tıklandığı tespit edilmekte ve 7. satırda, terfi edilecek olan taşın bulunduğu div alanına SqPromSelected class'ı eklenmektedir. Bu class'ın eklenmesi için aşağıdaki **SetSqPromSelected** fonksiyonu kullanılmaktadır;

```
1  function SetSqPromSelected(sq) {
2
3      if(GameController.BoardFlipped == BOOL.TRUE) {
4          sq = MIRROR120(sq);
5      }
6
7      $( ".SqProm" ).each(function( index ) {
8          if( (RanksBrd[sq] == 7 - Math.round($(this).position().top/60)) &&
           (FilesBrd[sq] == Math.round($(this).position().left/60)) ){
9              $(this).addClass('SqPromSelected');
10         }
11     });
12 }
```

SqPromSelected class'ı aşağıda görüldüğü gibi eklendiği alanının zemin rengini değiştirmektedir. Böylelikle o alandaki taşın seçilmiş olduğu anlaşılmaktadır;

```
.SqPromSelected{
   background-color:#216778;
}
```

PiecеProm class'ına ait bir alana tıklandığında çalıştırılan anonim fonksiyonun 8. ve 18. satırları arasında, beyaz taraf için bir döngü çalıştırılmaktadır. Bu döngü yardımıyla, terfi menüsünden seçilmiş olan taş tespit edilmekte ve uMv değişkenine yapılan hamle bilgisi atanmaktadır. Aynı zamanda terfi menüsüne ihtiyaç kalmadıysa (12. satır) bu menü tahtadan kaldırılmaktadır (13. satır). 19. ve 30. satırlar arasında aynı işlemler siyah taraf için yapılmaktadır. 32. satırda, **checkOppMoveAndPlay** fonksiyonu çağrılmaktadır. Bu fonksiyon yardımıyla karşı tarafın hamlesini yapıp yapmadığı kontrol edilmektedir. Şayet karşı taraf da hamlesini yapmışsa hamleler tahta üzerinde işlenmektedir. **checkOppMoveAndPlay** fonksiyonu makemove.js dosyası içerisinde aşağıdaki şekilde yer almaktadır;

```
1  function checkOppMoveAndPlay(){
2      if (uMv != NOMOVE){
3          TotalSeconds += parseInt(Increments);
4          console.log("TotalSeconds : " + TotalSeconds);
5          CountDownTimer.innerHTML = UpdateTimer(TotalSeconds);
6          noForward();
7
```

```
8       if(timing != 0){
9           if ( (player == 2 & oppMove == NOMOVE) ||
            (player == 1 & SearchController.best == NOMOVE) ){
10              TimerOn = BOOL.FALSE;
11          }
12      }
13
14      if(player == 1 && uMv != NOMOVE &&
        SearchController.best != NOMOVE){
15          MakeSimMove(uMv, SearchController.best);
16      }
17      if(player == 2){
18          socket.emit('move', {move:uMv, TotalSeconds:TotalSeconds} );
19          if ((uMv != NOMOVE) & ( oppMove != NOMOVE) ){
20              MakeSimMove(uMv, oppMove);
21          }
22      }
23
24  }
25 }
```

checkOppMoveAndPlay fonksiyonunun 2. satırında uMv'da bir hamle değeri olup olmadığı kontrol edilmektedir. Şayet geçerli bir hamle değeri varsa fonksiyondaki diğer işlemlere başlanmaktadır. 3. satırda, artış miktarı toplam kalan süreye eklenmektedir. 4. satırda, kalan süre konsola yazdırılmaktadır. 5. satırda, kullanıcı ara yüzündeki sayaç güncellenmektedir. 6. satırda, **noForward** fonksiyonu çağrılarak daha önceden yapılmış olan ileriye dönük hamleler silinmektedir. Örneğin; 6. hamleler yapıldıktan sonra oyuncu oyunu 4. hamleye kadar geri almış olsun. Oyuncu yeni bir 4. hamle belirledikten sonra 5. ve 6. hamlelerin silinmesi gerekmektedir. Çünkü artık oyuncu yeni bir hamle belirlemiş ve oyunun akışını değiştirmiştir. Şayet bir oyun süresi belirlenmişse (8. satır) ve karşı taraf henüz hamlesini yapmamışsa (9. satır) oyuncu geçerli bir hamle belirlediğinden dolayı oyuncunun sayacı durdurulmaktadır (10. satır). Şayet oyun tek oyunculu bir oyunsa ve her iki taraf da hamlelerini belirlemişse (14. satır) **MakeSimMove** fonksiyonu çağrılarak hamleler tahtaya işlenmektedir (15. satır). Şayet oyun iki oyunculu bir oyunsa (17. satır) oyuncunun hamlesi ve kalan süresi karşı tarafa iletilmek üzere sunucuya gönderilmektedir (18. satır). Her iki taraf da hamlesini belirlemişse (19. satır) **MakeSimMove** fonksiyonu çağrılarak hamleler tahtaya işlenmektedir. Sunucu "move" etiketli bir mesaj aldığında, mesajın etiketini "opponent move" olarak değiştirmekte ve data (move) kısmına

dokunmadan mesajı karşı tarafa iletmektedir. Bu işlem aşağıda görülmektedir (server.js);

```
1  socket.on('move', function(move){
2      socket.opponent.emit('opponent move', move);
3  });
```

Kullanıcı hamle yaparken bir oyun taşına tıklayabileceği gibi boş bir kareye de tıklayabilmektedir. Bu durumda aşağıdaki işlemler gerçekleştirilmektedir (gui.js);

```
1  $(document).on('click','.Square', function (e) {
2      if( moveDone == BOOL.FALSE &
       PromotionSelectionCalled == BOOL.FALSE ){
3          console.log('Square Click');
4          if(UserMove.from != SQUARES.NO_SQ) {
5              UserMove.to = ClickedSquare(e.pageX, e.pageY);
6              uPreMv = ParseMove(UserMove.from,UserMove.to);
7              if(uPreMv != PIECES.EMPTY){
8                  moveDone = BOOL.TRUE;
9
10                 if(PROMOTED(uPreMv) != PIECES.EMPTY){
11                     PromotionSelection(UserMove.from,UserMove.to);
12                 }else{
13                     uMv = uPreMv;
14                     checkOppMoveAndPlay();
15                 }
16             }
17             else{
18                 DeselectSq(UserMove.from);
19                 DeselectSq(UserMove.to);
20                 UserMove.from = SQUARES.NO_SQ;
21                 UserMove.to = SQUARES.NO_SQ;
22             }
23         }
24     }
25 });
```

Oyuncunun hamle yapmak için boş bir kareye tıklaması sonucu bir anonim fonksiyon çağrılmaktadır. Şayet hamle henüz tamamlanmamışsa ve terfi menüsü çağrılmamışsa fonksiyondaki diğer işlemlere geçilmektedir. 3. satırda, bir kareye tıklandığı mesajı konsola yazdırılmaktadır. Oyuncu hamlenin başladığı kareyi

(oyun taşını) belirlemişse (4. satır) 5. ve 15. satırlar arasındaki kodlar çalıştırılmaktadır. 5. satırda, tıklanılan yerin hangi kare olduğu tespit edilerek UserMove.to değişkenine atanmaktadır. 6. satırda yapılan hamlenin geçerli bir hamle olup olmadığı kontrol edilerek uPreMv değişkenine atanmaktadır. Şayet geçerli bir hamle yapılmışsa (7. satır) hamlenin tamamlandığını gösteren değişken aktif hale getirilmektedir (8. satır). Hamle sonucu bir terfi işlemi yapılması gerekiyorsa (10. satır) **PromotionSelection** fonksiyonu çağrılarak terfi menüsü tahta üzerinde oluşturulmaktadır (11. satır). Şayet bir terfi işlemi yoksa (12. satır) kullanıcı hamlesi kesinleştirilmekte (13. satır) ve **checkOppMoveAndPlay** fonksiyonu yardımıyla hamlelerin tahtaya işlenmesi sağlanmaktadır. Şayet yapılan hamle geçerli bir hamle değilse (17. satır) 18. ve 21. satırlar arasındaki değerler sıfırlanarak geçerli bir hamle yapılması beklenmektedir.

İki oyunculu oyunlarda, rakibin hamlesini belirlediğini gösteren "opponent move" mesajı istemci tarafından alındığında aşağıdaki işlemler yapılmaktadır (makemove.js);

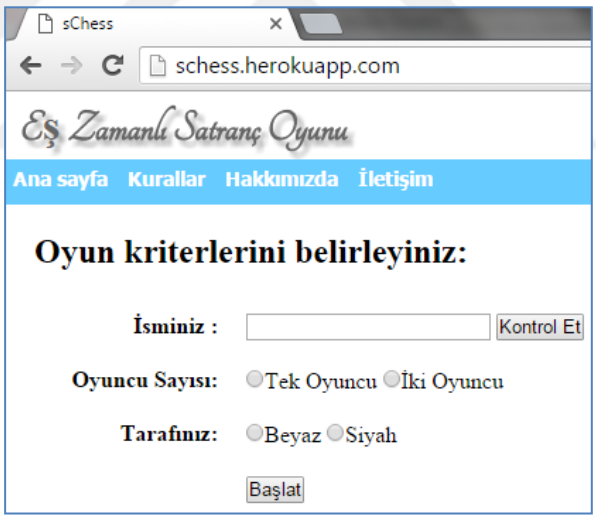
```
 1  socket.on('opponent move', function(data){
 2      oppMove = data.move;
 3      TotalSecondsOpp = data.TotalSeconds;
 4      CountDownTimerOpp.innerHTML = UpdateTimer(TotalSecondsOpp);
 5      if(uMv == NOMOVE){
 6          TimerOnOpp = BOOL.FALSE;
 7      }else if(uMv != NOMOVE){
 8          if (PROMOTED(uMv) != PIECES.EMPTY){
 9              ClearAllProm();
10          }
11          MakeSimMove(uMv, oppMove);
12      }
13  });
```

Alınan "opponent move" etiketli mesajla birlikte gönderilmiş olan veri (rakibin hamle ve sayaç bilgisi) çalıştırılan anonim fonksiyona parametre olarak verilmektedir (1. satır). 2. satırda, rakibin hamle bilgisi oppMove değişkenine atanmaktadır. 3. satırda rakibin kalan süresi TotalSecondsOpp değişkenine atanmaktadır. Böylelikle rakip için tutulan sayaçta bir sapma varsa düzeltilmektedir. 4. satırda, kullanıcı ara yüzünde gösterilen rakibin sayaç bilgisi güncellenmektedir. Şayet oyuncu kendi hamlesini belirlememişse (5. satır), rakibi hamlesini belirlediği için rakibinin sayacını durdurmaktadır (6. satır). Şayet

oyuncu da hamlesini belirlemişse (7. satır) 8. ve 12. satırlar arasındaki kod çalıştırılmaktadır. Oyuncunun belirlediği hamle sonucu bir terfi işlemi varsa (8. satır) terfi işlemi için oluşturulmuş olan menü tahtadan silinmektedir (9. satır). Her iki tarafın hamlesi belirlenmeden hamleler tahtada gösterilmediği için karşı tarafın hamlesi belirlenene kadar terfi menüsü ve yapılan seçim tahta üzerinde gösterilmektedir. 11. satırda, **MakeSimMove** fonksiyonu yardımıyla hamleler işleme alınmaktadır.

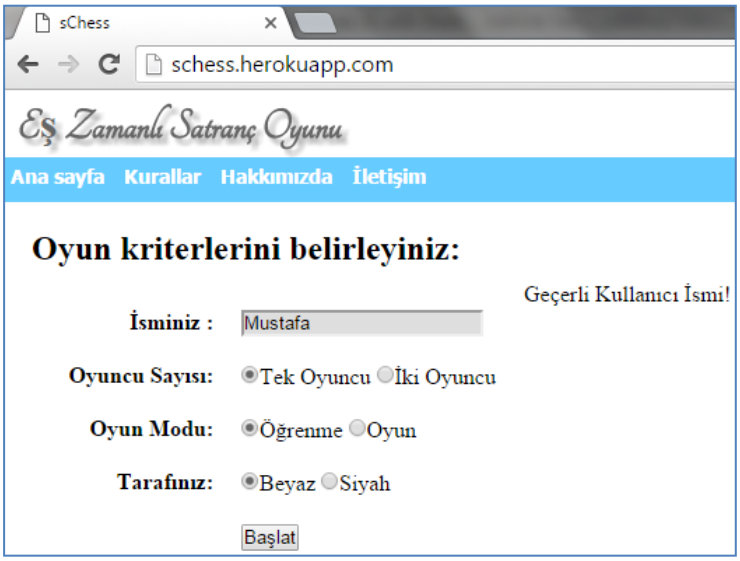
5 EŞ ZAMANLI SATRANÇ UYGULAMASININ KULLANIMI

Uygulamaya ait web ara yüzüne "**http://schess.herokuapp.com/**" adresinden veya güvenli (Public Key Infrastructure (Açık Anahtar Altyapısı) - PKI) bir erişim sağlayan "**https://schess.herokuapp.com/**" adresinden erişilebilmektedir. İlgili web adresine erişildiğinde Şekil 5-1'deki ana sayfa karşılaşılmaktadır. Bu alanda oynamak istenilen eş zamanlı satranç oyununun kriterleri belirlenmektedir. On iki karakteri geçmeyecek şekilde bir isim belirlenerek "İsminiz : " kısmının karşısındaki alana (Text Box) yazılmaktadır. Ardından da "Kontrol Et" düğmesine basılarak, fare ile ekranda boş bir yere tıklanarak veya Tab tuşuna basılarak girilen ismin daha önce alınıp alınmadığı kontrol edilmektedir. Belirlenen isim hali hazırda kullanılıyorsa o ismin kullanıldığına dair bir uyarı mesajı çıkmaktadır ve yeni bir isim girilmesi istenmektedir. Geçerli bir isim belirlendikten sonra "Oyuncu Sayısı" kısmına geçilmektedir.



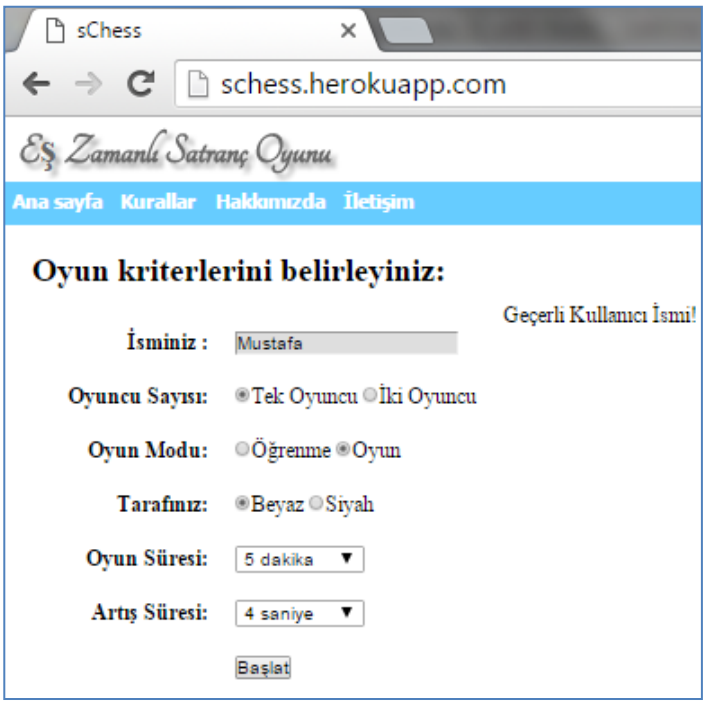
Şekil 5-1 Eş Zamanlı Satranç Ana Sayfa

"Oyuncu Sayısı" kısmında bilgisayara karşı (Tek Oyuncu) mı yoksa başka bir oyuncuya (İki Oyuncu) karşı mı oynanacağı belirlenmektedir. Tek Oyuncu seçeneği seçildiğinde, menüde "Oyun Modu" kısmı belirmektedir (Şekil 5-2). "Öğrenme" modunda, oyun esnasında hamleyi geri alma, ileri alma gibi seçeneklere izin verilmekte fakat "Oyun" modunda ise bu seçeneklere izin verilmemektedir.



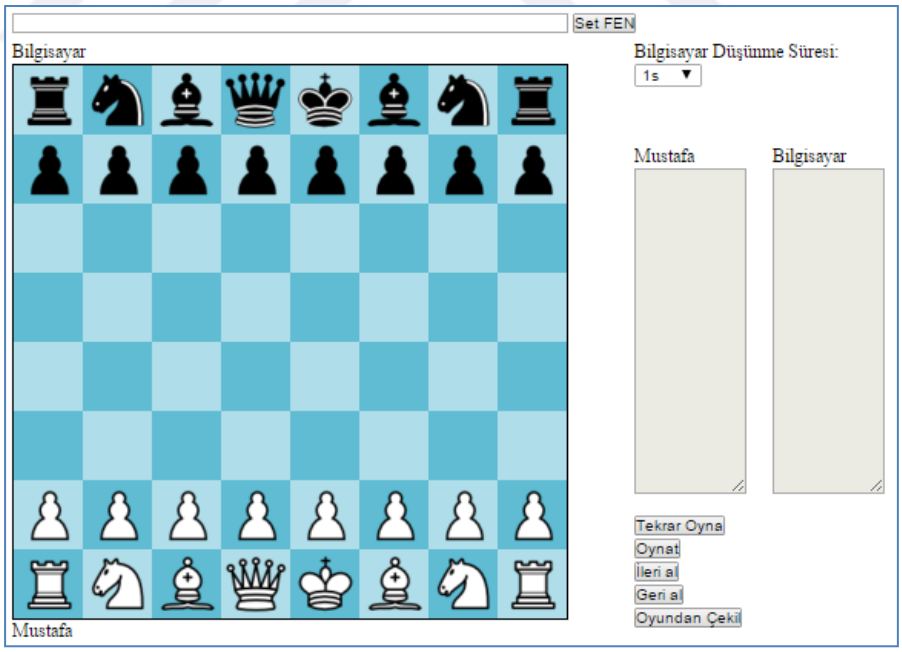
Şekil 5-2 Oyun Kurma: Tek Oyuncu Öğrenme Modu

"Oyun" modu seçildiğinde menüde "Oyun Süresi" ve "Artış Süresi" kısımları belirmektedir (Şekil 5-3). Oyun Süresi kısmında oyunun maksimum ne kadar süreyle oynanacağı belirlenmektedir. Herhangi bir sınır konulmayacaksa "Sınırsız" seçeneği seçilmektedir. Artış Süresi kısmında ise her bir geçerli hamle sonucunda mevcut süreye ne kadar ekleme yapılacağı belirlenmektedir. Herhangi bir ekleme yapılmayacaksa "Artışsız" seçeneği seçilmektedir.



Şekil 5-3 Oyun Kurma: Tek Oyuncu Oyun Modu

Tek kullanıcı seçeneği ve Öğrenme modu seçilip Başlat düğmesine basıldığında Şekil 5-4'deki ekran görüntüsü görünmektedir. Set FEN tuşunun sol tarafına bir FEN metni eklenip Set FEN düğmesine basıldığında tahtadaki taşlar FEN dizisine göre yerleştirilecektir (Bkz.3.8). Bu kısım hemen altında sırasıyla rakibin ismi, oyun tahtası ve oyuncunun kendi ismi yer almaktadır. "Bilgisayar Düşünme Süresi" kısmında bilgisayarın ne kadar süreyle en iyi hamleyi arayacağı belirlenmektedir. Sürenin uzun tutulması bilgisayarın daha iyi bir hamle belirlemesini sağlamaktadır. Bu kısmın altında yer alan ve üzerlerinde oyuncuların isimlerinin yazılı olduğu metin kutucukları ise oyun esnasında kullanıcıların hangi hamleleri yaptığını gösteren alanlardır. "Tekrar Oyna" düğmesi ile hali hazırda belirlenmiş olan kriterler kullanılarak oyun baştan başlatılmaktadır. "Oynat" düğmesine basıldığında bilgisayar hem siyah hem beyaz taraf için en iyi hamleyi bulmakta ve hamleleri uygulamaktadır. "Geri al" düğmesi ile oyun bir önceki pozisyona getirilmektedir (k mevcut tahta durumu olmak üzere k = k-1 yapılmaktadır). "İleri al" düğmesi ile oyun bir sonraki pozisyona getirilmektedir (n son tahta durumu ve k mevcut tahta durumu olmak üzere k < n şartıyla k+1 yapılır). "Oyundan Çekil" düğmesi kullanılarak yenilgi kabul edilerek daha fazla oynanılmayacağı belirtilmektedir.

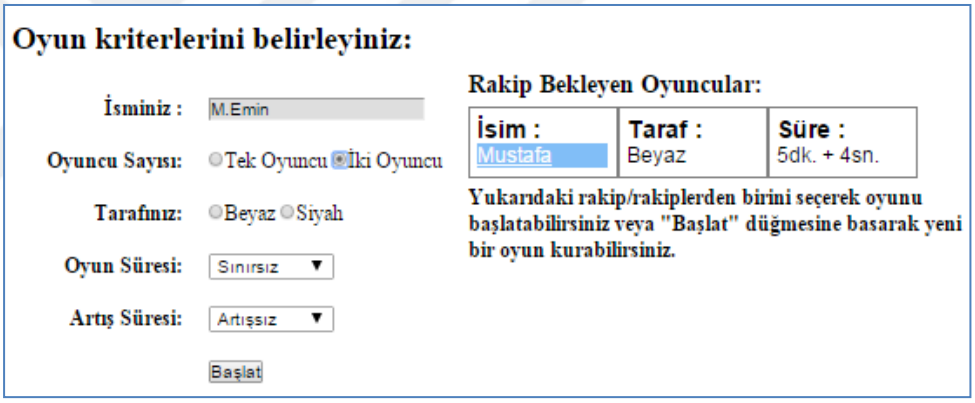


Şekil 5-4 Tek Kullanıcı: Öğrenme Modu

Tek kullanıcı Oyun seçeneğinde "set FEN", "Bilgisayar Düşünme Süresi", "Oynat", "İleri al" ve "Geri al" seçenekleri kaldırılmıştır. "Tekrar Oyna" butonu ise oyun sonunda aktif olmaktadır.

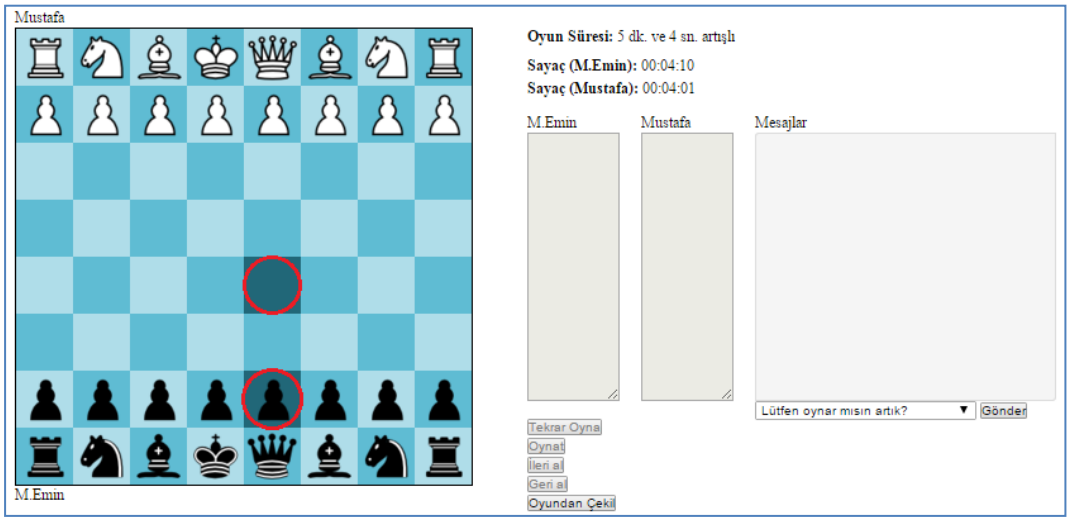
Mevcut oyun kriterlerinden faklı bir oyun kurmak için menüden "Ana Sayfa" kısmına tıklanarak ana menüye dönülmektedir.

"İki Oyuncu" seçeneği kullanılarak gerçek bir oyuncuyla oynanmak istendiğinde bir tarafın oyun kriterlerini belirlemesi diğer tarafın da belirlenen oyun kriterlerini kabul ederek oyuna dahil olması gerekmektedir. Oyunu kuran kişi seçenekleri belirleyip başlat düğmesine bastığında " Yeni bir oyun kurdunuz. Bir rakip oyuncunun oyuna katılması bekleniyor..." mesajını görmektedir. Rakip oyuncu "İki Oyuncu" seçeneğini seçtiğinde Şekil 5-5'deki ekran görüntüsü ile karşılaşmaktadır. Bu durumda rakip oyuncunun iki farklı seçeneği bulunmaktadır. Ya kendisi de tarafını, oyun süresini ve artış süresini belirleyerek başlat düğmesine basıp yeni bir oyun kuracak ya da "Rakip Bekleyen Oyuncular" listesinde bulunan oyunculardan birinin isminin üzerinde tıklayarak o oyuncunun belirlediği kriterleri kabul edecek ve o şartlarda oyuna başlayacaktır.



Şekil 5-5 İki Oyuncu Seçeneği: Oyun Kurulumu

Oyun başladığında Şekil 5-6'daki ekran çıktısı belirmektedir. Şekil 5-6'daki görüntü siyah tarafa ait görüntüdür. Tahtanın alt tarafında oyuncunun kendisine, üst tarafında ise rakip oyuncuya ait taşlar yerleştirilmektedir. Bir taşı hareket ettirmek için hareket ettirilecek taşın üzerinde tıklanmakta ardından da hareket edilecek kareye tıklanmaktadır. Belirlenen toplam oyun süresi ve artış miktarı "Oyun Süresi" kısmında görülmektedir. Kullanıcılara ait "Sayaç"larda ise kullanıcının ne kadar süresi kaldığı görülmektedir. Sayacı sıfırlanan taraf oyunu kaybetmektedir. Sayaçlar aynı anda sıfırlandığında oyun berabere sonuçlanmaktadır. İki Oyuncu seçeneğinde, Tek Oyuncu seçeneğinden farklı olarak tarafların kısıtlı da olsa haberleşebilmesi için Mesajlar kısmı bulunmaktadır. Bu kısımda yazılı kısmı kullanıcının girmesine müsaade edilmeme sebebi yasal sorumluluktan kaçınılmasıdır.



Şekil 5-6 İki Oyuncu Seçeneği Oyun Ekranı

6 SONUÇLAR

Bu tez çalışmasıyla, satranç oyunundaki, ilk hamle avantajının başlayan tarafa sağladığı üstünlüğün ortadan kaldırılması amacıyla geliştirilen kurallar setinin tutarlılığı test edilmiş ve kuralların tutarlı olduğu görülmüştür. Ancak oyunun daha etkin bir şekilde oynanabilmesi için tez önerisinde 4. kural olarak verilen kuralın ilk cümlesi korunmuş ve geri kalan kısmı değiştirilmiştir. Şayet tez önerisinde belirtildiği gibi hamlelerin karşılıklı olarak geri alınma kuralı korunmuş olsaydı, dezavantajlı olan tarafa üçlü tekrar kuralını kullanma şansı doğacak ve böylelikle adaletsiz bir durum oluşacaktı. Oluşan bu durum da tezin amaçlarından biri olan adil oyun düşüncesine aykırı bir sonuca sebep olacaktı. İstenmeyen bu durumun ortaya çıkmaması için kural değişikliği yoluna gidilmiştir.

Tez önerisinde 4. kural olarak verilen kural aşağıdaki gibidir;

Siyah ve Beyaz'ın eşzamanlı olarak yapacakları bir hamle sonucu iki taş Satranç tahtasının aynı karesine gelirse bu taşların ikisi de oyun dışı kalır (karşılıklı olarak birbirlerini yerler). Aynı kareye gelen taşlardan birisi Şah ise hamle karşılıklı olarak geri alınır. Geri almadan bir sonraki hamle çiftinde taraflar farklı hamleler yapabilecekleri gibi yine aynı kareye hamle yapabilirler. Bu durum toplam üç kez tekrarlanırsa oyun Pat olur.

Bu kural aşağıdaki şekilde değiştirilmiştir;

Siyah ve Beyaz'ın eşzamanlı olarak yapacakları bir hamle sonucu iki taş satranç tahtasının aynı karesine gelirse bu taşların ikisi de oyun dışı kalmaktadır (karşılıklı olarak birbirlerini yerler). Aynı kareye gelen taşlardan birisi şah diğeri piyon ise FIDE kuralları gereği şah tahtadan çıkmadığı için sadece piyon tahtadan çıkarılmaktadır (Oyun kuralları gereği şahın, piyon haricinde başka bir taşla aynı kareye düşme ihtimali bulunmamaktadır.).

Bu tez çalışmasında aynı zamanda Eşzamanlı Hamle Tabanlı Satranç oyunu için GUI tabanlı bir Web uygulaması geliştirilmiştir. Bu Web uygulaması ile etkileşimli, gerçek zamanlı insan – insan ve insan - bilgisayar eşzamanlı hamle tabanlı satranç oynamanın mümkün olduğu prototip bir yapı oluşturulmuştur.

Bu tez çalışması sonucunda yeni araştırma konuları da ortaya çıkmıştır. Bunlar;

1. Eş zamanlı satranç oyunu için en iyi hamleyi arama algoritması geliştirme ve uygulanması,
2. Eş zamanlı satranç oyunu için tahta pozisyonlarının yeniden değerlendirilmesi çalışması,
3. Eş zamanlı satranç oyunu için Perft(Performance Test) çalışmasıdır.

7 KAYNAKLAR DİZİNİ

Banks G., S. C., Banks R. (FRC only), Szots G. and Russell N. (2016, 9 Ocak 2016). "CCRL (Computer Chess Rating Lists)." Retrieved 12 Ocak 2016, 2016, from http://www.computerchess.org.uk/ccrl/404/.

Bluefever-I. (2014, 23 Haziran 2014). "Chess Engine In C." Retrieved 18 Ocak 2016, from https://www.youtube.com/playlist?list=PLZ1QII7yudbc-Ky058TEaOstZHVbT-2hg.

Bluefever-II. (2014, 25 Haziran 2014). "Programming A Javascript Chess Engine." Retrieved 18 Ocak 2016, from https://www.youtube.com/playlist?list=PLZ1QII7yudbe4gz2gh9BCI6VDA-xafLog.

chess-programming-I. (2016). "Chess Engine Communication Protocol." from http://chessprogramming.wikispaces.com/Chess+Engine+Communication+Protocol.

chess-programming-II. (2016). "Getting Started." Retrieved 21 Ocak 2016, from http://chessprogramming.wikispaces.com/Getting+Started.

Mann, M. (2009). "Chess Engine Communication Protocol." Retrieved 18 Ocak 2016, from http://home.hccnet.nl/h.g.muller/engine-intf.html.

Warkentin-I, J. (2014, 28 Haziran 2014). "Programming a Simple Java Chess Engine - Logic Crazy." Retrieved 18 Ocak 2016, from https://www.youtube.com/playlist?list=PLQV5mozTHmaffB0rBsD6m9VN1azgo5wXl.

Warkentin-II, J. (2014, 21 Temmuz 2014). "Programming an Advanced Java Chess Engine - Logic Crazy." Retrieved 18 Ocak 2016, from https://www.youtube.com/playlist?list=PLQV5mozTHmacMeRzJCW_8K3qw2miYqd0c.

8 ÖZGEÇMİŞ

Mustafa TÜRK, 18 Eylül 1978'de Kırşehir'de doğmuştur. Türkiye Cumhuriyeti vatandaşıdır. Lisans eğitimini 1996 2001 yılları arasında Anadolu Üniversitesi İktisadi ve İdari Bilimler Fakültesi Maliye Bölümünde tamamlamıştır. 2009-2010 yılları arasında Sakarya Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar ve Bilişim Mühendisliği dalında yüksek lisans eğitimini tamamlamıştır. 2012'dan bu yana Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü yüksek lisans öğrencisidir. 1996 senesinden beri TSK'de bilişim alanında çalışmaktadır. 2002 yılında Kosova Barış Görev Kuvveti(KFOR - Kosova) ve 2009 yılında Uluslararası Güvenlik Destek Gücü'nde (ISAF - Afganistan) sistem yöneticisi olarak görev yapmıştır. Halen NATO MEBS Grup Komutanlığı'nda ağ yöneticisi olarak çalışmaktadır.