T.C
İSTANBUL AYDIN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

HAPİS TAVLASI (BACKGAMMON PRISON) OYUN YAZILIMININ GELİŞTİRİMİ

YÜKSEK LİSANS TEZİ

Hazırlayan

Hakan ŞAHİN

Tez Danışmanı

Doç. Dr. Ahmet BABANLI

İSTANBUL-2013

T.C
İSTANBUL AYDIN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

HAPİS TAVLASI (BACKGAMMON PRISON) OYUN YAZILIMININ GELİŞTİRİMİ

YÜKSEK LİSANS TEZİ

Hakan ŞAHİN

Y1013.010008

Tez Danışmanı

Doç. Dr. Ahmet BABANLI

İSTANBUL-2013

KABUL VE ONAY BELGESİ

Doç. Dr. Ahmet BABANLI'nın danışmanlığında **Hakan ŞAHİN** tarafından hazırlanan **"Hapis Tavlası (Backgammon Prison) Oyun Yazılımının Geliştirimi"** adlı bu çalışma, jürimiz tarafından İstanbul Aydın Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans Tezi olarak kabul edilmiştir.

…../…../2013

JURİ

Danışman : Doç.Dr. Ahmet BABANLI ……

Üye : Prof.Dr. Ali GÜNEŞ ……

Üye : Yard.Doç.Dr. Metin ZONTUL ……

Tezin Savunulduğu Tarih: …../…../2013

Bu tez çalışması İstanbul Aydın Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararı ile onaylanmıştır.

Fen Bilimleri Enstitüsü Müdürü

ÖNSÖZ

Makinelerin karşılaştıkları ve daha önceden görmemiş oldukları sorunları çözmesini amaçlayan Yapay Zeka, ilk kez 1956 yılının yaz aylarında Darmouth Üniversitesi'nde yapılan bir konferansta ortaya çıkmıştır. İlk yıllarında çok hızlı bir ivme yakalayan ve pek çok insanı hayrete düşüren yapay zeka programları, ilerleyen yıllarda aynı ivmeyi ne yazık ki koruyamamıştır. Bunun nedeni, insanların kolaylıkla çözdüğü sorunları bir makineye anlatmanın zorluğu ve makinelerin o zamanki işlem gücü kısıtlarıydı. Ne var ki, son yirmi yıldır çok yüksek bir hızla ilerleyen teknoloji, karmaşık yapay zekaya olanak sağlayacak donanımların üretilmesini sağlamış ve modern yapay zekanın ilerlemesinde büyük katkıda bulunmuştur. Günümüzde, en kolay ve en etkin olarak programlanabilen aygıtlar bilgisayarlar olduğu için yapay zeka bir bilgisayar bilimi olmuştur ve halen pek çok uygulamada etkin olarak kullanılmaktadır.

TEŞEKKÜR

Bu tezin gerçekleştirilmesinde, başlangıcından sonuna kadar, gerekli bütün yardım, tavsiye ve yönlendirmeleri yapan, karşılaştığım problemlerin çözümünde deneyimlerinden yararlandığım sayın hocam Doç. Dr. Ahmet BABANLI'ya katkılarından dolayı teşekkür ederim.

Bu çalışmam süresince bana her zaman destek olarak, tecrübeleriyle bana yol gösteren değerli hocalarım Prof. Dr. Ali GÜNEŞ'e ve Yard. Doç. Dr. Metin ZONTUL'a teşekkür ederim.

Tüm eğitim hayatım boyunca benden maddi manevi hiçbir yardımı esirgemeyen canım aileme ve hayatımın güzelliği Canan ÇOBAN'a en içten teşekkürlerimi ve şükranlarımı sunarım.

İÇİNDEKİLER

KISALTMALAR

GA **:** Genetik Algoritma

KSO **:** Kedi Sürüsü Optimizasyonu

PSO **:** Parçacık Sürü Optimizasyonu

YZ **:** Yapay Zeka

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

GİRİŞ

Oyun programlamada kullanılan yapay zekânın amacı oyuncu ile etkileşimde bulunan karakterlerin ya da oyunun geçtiği ortamın mümkün olduğunca gerçek insan, topluluk ve dünya (ya da kimyasal- fiziksel - biyolojik olarak anlamlı ortam) yaşam ortamına benzetilmeye çalışılmasıdır. Başka bir anlamda da bilgisayarın yetenekli bir oyuncuya aynı derecede yetenekli karşılık vermesi, oyuncunun kurduğu planlara benzeyen planlar ya da tuzaklar kurması, gerektiğinde oyuncuyu zor duruma düşürüp onu yenebilmesi oyun programındaki yapay zekâ kalitesini belirler [1].

Bazen tek başlarına hiçbir iş yapamayan varlıklar, toplu hareket ettiklerinde çok zekice davranışlar sergileyebilmektedir. Bir topluluğa ait bireyler, en iyi bireyin davranışından ya da diğer bireylerin davranışlarından ve kendi deneyimlerinden yararlanarak yorum yapmakta ve bu bilgileri ileride karşılaşacakları problemlerin çözümleri için bir araç olarak kullanmaktadırlar. Örneğin, bir canlı sürüsünü oluşturan bireylerden birisi bir tehlike sezdiğinde bu tehlikeye karşı tepki verir ve bu tepki sürü içinde ilerleyip tüm bireylerin tehlikeye karşı ortak bir davranış sergilemesini sağlar. Canlıların sürü içerisindeki bu hareketleri gözlemlenerek sürü zekâsı tabanlı optimizasyon algoritmaları geliştirilmiştir (Murty, 2003).

Bu çalışmada sezgisel sürü optimizasyon algoritmaları anlatılmış ve bu algoritmalardan yapay balık sürü algoritması ve ateş böceği sürü optimizasyonundan faydalanılarak hapis tavlası oyunu ele alınmıştır.

Birinci bölümde sezgisel optimizasyondan, sezgisel optimizasyonun gruplarından, sürü zekâsı optimizasyon algoritmalarından, kedi sürüsü optimizasyonlarından ve yapay arı koloni algoritmasından bahsedilmiştir.

İkinci bölümde sezgisel problem örneklerinden örnekler verilmiştir.

Üçüncü bölümde oyunlarda sezgisel çözümlerden örnekler verilmiştir.

Dördüncü bölümde k hapis tavlası oyunu ve kurallarından, sezgisel sürü optimimazyon algoritmalarının mevcut hapis tavlası probleminde nasıl kullanıldığından, problemin durum uzayı ve hapis tavlasının oyun ağacından, hapis tavlası oyununda kullanılan ve oyun kuralları doğrultusunda oluşturulan fonksiyonlara ve genel program akışına yer verilmiştir. Beşinci bölümde ise yapılan bu çalışmanın sonuç kısmına yer verilmiştir.

BÖLÜM I

SEZGISEL YAPAY ZEKA

1.1 Yapay Zeka Nedir?

Yapay zeka konusuna girmeden önce zekanın kısa bir tanımını yapmak yerinde olacaktır. Zeka, insanın düşünme, akıl yürütme, nesnel gerçekleri algılama, kavrama, yargılama, sonuç çıkarma, soyutlama, öğrenme yeteneklerinin tümüdür. Ayrıca soyutlama, öğrenme ve yeni durumlara uyma gibi yetenekler de zeka kapsamı içindedir (Nabiyev, 2003).

Yapay zeka ise, bu özelliklere sahip organik olmayan sistemlerdeki zekadır. Yapay zeka kabaca; bir bilgisayarın ya da bilgisayar denetimli bir makinenin, genellikle insana özgü nitelikler olduğu varsayılan akıl yürütme, anlam çıkartma, genelleme ve geçmiş deneyimlerden öğrenme gibi yüksek zihinsel süreçlere ilişkin görevleri yerine getirme yeteneği olarak tanımlanmaktadır (Nabiyev, 2003).

Yapay zeka dört kategoriye ayrılabilmektedir:

- İnsan gibi düşünen sistemler
- İnsan gibi davranan sistemler.
- Mantıklı düşünen sistemler.
- Mantıklı davranan sistemler.

İnsan gibi davranan sistemler

Yapay zeka araştırmacılarının baştan beri ulaşmak istediği ideal, insan gibi davranan sistemler üretmektir. Turing zeki davranışı, bir sorgulayıcı kandıracak kadar bütün bilişsel görevlerde insan düzeyinde başarım göstermek olarak tanımlamıştır. Bunu ölçmek için de Turing Testi olarak bilinen bir test önermiştir. Turing testinde denek sorgulayıcıyla bir terminal aracılığıyla haberleşir. Eğer sorgulayıcı deneğin insan mı yoksa bir bilgisayar mı olduğunu anlayamazsa denek Turing testini geçmiş sayılır. Turing, testini tanımlarken zeka için bir insanın fiziksel benzetiminin gereksiz olduğunu düşündüğü için sorgulayıcıyla bilgisayar arasında doğrudan fiziksel temasdan söz etmekten kaçınmıştır. Burada vurgulanması gereken nokta, bilgisayarda zeki davranışı

üreten sürecin insan beynindeki süreçlerin modellenmesiyle elde edilebileceği gibi tamamen başka prensiplerden de hareket edilerek üretilmesinin olası olmasıdır [2].

İnsan gibi düşünen sistemler

İnsan gibi düşünen bir program üretmek için insanların nasıl düşündüğünü saptamak gerekir. Bu da psikolojik deneylerle yapılabilir. Yeterli sayıda deney yapıldıktan sonra elde edilen bilgilerle bir kuram oluşturulabilir. Daha sonra bu kurama dayanarak bilgisayar programıüretilebilir. Eğer programın giriş/çıkış ve zamanlama davranışı insanlarınkine eşse programın düzeneklerinden bazılarının insan beyninde de mevcut olabileceği söylenebilir. İnsan gibi düşünen sistemler üretmek bilişsel biliminin araştırma alanına girmektedir. Bu çalışmalarda asıl amaç genellikle insanın düşünme süreçlerini çözümlemede bilgisayar modellerini bir araç olarak kullanmaktır [2].

Rasyonel düşünen sistemler

Bu sistemlerin temelinde mantık yer alır. Burada amaç çözülmesi istenen sorunu mantıksal bir gösterimle betimledikten sonra çıkarım kurallarını kullanarak çözümünü bulmaktır. YZ'de çok önemli bir yer tutan mantıkçı gelenek zeki sistemler üretmek için bu çeşit programlar üretmeyi amaçlamaktadır. Bu yaklaşımı kullanarak gerçek sorunları çözmeye çalışınca iki önemli engel karşımıza çıkmaktadır. Mantık formel bir dil kullanır. Gündelik yaşamdan kaynaklanan, çoğu kez de belirsizlik içeren bilgileri mantığın işleyebileceği bu dille göstermek hiç de kolay değildir. Bir başka güçlük de en ufak sorunların dışındaki sorunları çözerken kullanılması gerekecek bilgisayar kaynaklarının üstel olarak artmasıdır [2].

Rasyonel davranan sistemler

Amaçlara ulaşmak için inançlarına uygun davranan sistemlere rasyonel denir. Bir ajan algılayan ve harekette bulunan bir şeydir. Bu yaklaşımda YZ rasyonel ajanların incelenmesi ve oluşturulması olarak tanımlanmaktadır. Rasyonel bir ajan olmak için gerekli koşullardan biri de doğru çıkarımlar yapabilmek ve bu çıkarımların sonuçlarına göre harekete geçmektir.

Yapay zekanın temelleri bir çok farklı alanlardan beslenmektedir. Felsefe, Matematik, Algoritma, Ekonomi, Psikoloji, Bilgisayar Mühendisliği, Sinir bilimleri, Kontrol teorisi ve Sibernetik ve Dilbilim başlıcaları olarak sayılabilmektedir (Russell & Norvig, 2003).

1.3 Yapay Zekanın Amaçları

Yapay zeka alanında yapılan çalışmalarda amaçları şöyle sıralayabiliriz:

1) İnsan beyninin fonksiyonlarını bilgisayar modelleri yardımıyla anlamaya çalışmak.

2) İnsanların sahip olduğu zihinsel yetenekleri, bilgi kazanma, öğrenme ve buluş yapmada uyguladıkları strateji, metot ve teknikleri araştırmak.

3) Bu öğrenme metotlarını formel hale getirmek ve bilgisayarlarda bilgi sistemleri halinde uygulamak.

4) İnsanlarını bilgisayar kullanımını kolaylaştıracak insan/bilgisayar ara birimleri geliştirmek.

5) Belli bir uzmanlık alanı içindeki bilgileri bir 'bilgi sistemi' (veya 'uzman sistem') halinde toplamak.

6) Geleceğin bilgi toplumunun kurulmasında önemli rol oynayacak 'genel bilgi sistemleri' geliştirmek.

7) Yapay zeka iş yardımcıları ve 'zeki robot timleri' geliştirmek.

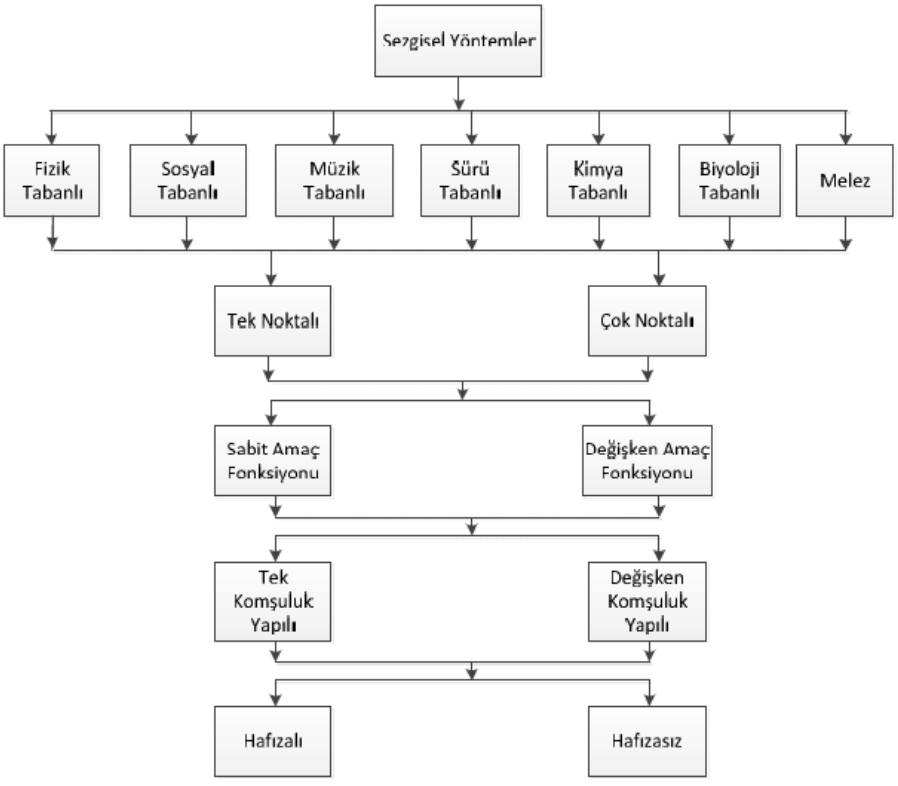
8) Bilimsel araştırma ve buluşlarda faydalanmak üzere, 'araştırma yardımcıları' geliştirmek [3].

1.4 Sezgisel Optimisazyon

Sezgisel algoritmalar, herhangi bir amacı gerçekleştirmek veya hedefe varmak için doğal fenomenlerden esinlenen algoritmalardır. Bu algoritmaların, çözüm uzayında optimum çözüme yakınsaması ispat edilememektedir. Yani sezgisel algoritmalar yakınsama özelliğine sahip olmaktadır, ama kesin çözümü garanti edememektedir ve bu kesin çözümün yakınlarında bir çözüm garanti edebilmektedir. Anlaşılırlık yönünden sezgisel algoritmaların karar verici açısından çok daha basit olabilmesinden, optimizasyon problemlerinin

kesin çözümü bulma işleminin tanımlanamadığı bir yapıya sahip olmasından ve öğrenme amaçlı ve kesin çözümü bulma işleminin bir parçası olarak kullanabilirliğinden sezgisel algoritmalara ihtiyaç duyulmaktadır (Karaboğa, 2011).

Genel amaçlı sezgisel yöntemler; biyoloji tabanlı, fizik tabanlı, sürü tabanlı, sosyal tabanlı, müzik tabanlı ve kimya tabanlı olmak üzere altı farklı grupta değerlendirilmektedir. Ayrıca bunların birleşimi olan melez yöntemler de vardır. Bahsedilen bu yöntemler Şekil-1'de sunulmaktadır. Genetik algoritma (GA), diferansiyel gelişim algoritması, karınca koloni algoritmaları, yapay sinir ağları, arı koloni algoritmaları ve yapay bağışıklık sistemleri biyolojik tabanlı; emperyalist yarışmacı algoritma, parlamenter optimizasyon algoritması ve tabu arama sosyal tabanlı; yapay kimyasal reaksiyon algoritması kimya tabanlı; armoni arama algoritması müzik tabanlı; ısıl işlem, büyük patlama büyük sıçrama, yer çekimsel arama algoritması, merkez kuvvet optimizasyonu, zeki su damlacıkları algoritması ve elektromanyetizma algoritması fizik tabanlı ve Parçacık Sürü Optimizasyonu (PSO), KSO sürü tabanlı algoritma ve modellerdir. Kültürel algoritma da hem biyoloji hem de sosyal tabanlı algoritma olarak sınıflandırılabilir (Alataş, 2007).



Şekil-1: Sezgisel Yöntemler

1.5 Sürü Zekası Optimisazyon Algoritmaları

Sürü, birbirleriyle etkilesen dağınık yapılı bireyler yığını anlamında kullanılır. Bireyler insan veya karınca olarak ifade edilebilir. Sürülerde N adet temsilci bir amaca yönelik davranışı gerçekleştirmek ve hedefe ulaşmak için birlikte çalışmaktadır. Kolaylıkla gözlenebilen bu "kollektif zekâ" temsilciler arasında sık tekrarlanan davranışlardan doğmaktadır. Temsilciler faaliyetlerini idare etmek için basit bireysel kurallar kullanmakta ve grubun kalan kısmıyla etkileşim yolu ile sürü amaçlarına ulaşmaktadır. Grup faaliyetlerinin toplamından bir çeşit kendini örgütleme doğmaktadır.

Aşağıda şimdiye kadar farklı araştırmacıların önerdiği sürü zekâsı optimizasyon algoritmaları alt başlıklar halinde açıklanmıştır (Akyol, 2012).

1.5.1 Ateş Böceği Algoritması

Ateş böceği algoritması, Dr. Xin-She Yang (Cambridge Üniversitesi-2007) tarafından geliştirilen ve tropikal iklim bölgelerindeki ateşböceklerinin sosyal davranışlarını baz alan bir meta sezgisel optimizasyon algoritmasıdır (Yang, 2009). Bu algoritma diğer sürü zekâsı tabanlı algoritmalarla birçok benzerliği bulunmasına rağmen kavram ve uygulamada daha basittir. Bir ateş böceğinin ışıklarını yakıp söndürmesinin birincil amacı, diğer ateş böceklerini çekmek için bir sinyal sistemi olarak hareket etmektir. Yanıp sönen ışıkların üretimindeki karmaşık biyokimyasal sürecin detayları ve gerçek amacı bilim dünyasında hâlâ bir tartışma konusu olmasına rağmen, birçok araştırmacı yanıp sönen ışıkların ateşböceğine, arkadaşlarını bulmada, olası avlarını çekmede ve avcılarından kendilerini korumada yardımcı olduğuna inanmaktadır.

Ateşböceği algoritmasında, verimli optimal çözümler elde etmek için, verilen bir optimizasyon probleminin amaç fonksiyonu, ateşböceği sürüsüne parlak ve daha çekici yerlere gitmede yardım eden yanıp sönen ışık ya da ışık şiddeti ile ilişkili olmaktadır. Bütün ateş böcekleri tek cins olarak kabul edilmektedir ve birbirilerini çekmeleri bu algoritmanın temelini oluşturmaktadır. Bir ateş böceği ne kadar parlak olursa diğer ateş böcekleri için o kadar çekici hale gelmektedir. Kendisinden daha parlak bir ateşböceği gördüğünde ona doğru gidecektir (Apostolopoulos, 2011, Yang, 2009).

1.5.2 Ateş Böceği Sürü Optimizasyonu

Ateş böceği sürü optimizasyonu, K. N. Krishnanand ve D. Ghose tarafından 2005 yılında geliştirilmiştir (Krishnanand, 2005). Çok modelli fonksiyonları optimize etmek için önerilen sürü zekâsı tabanlı bir algoritmadır. Bu yöntemi kullanmanın temel amacı tüm yerel maksimumları yakalamayı sağlamaktır. Çok modelli fonksiyon optimizasyon problemlerinde, ateş böceği sürü optimizasyonu ve önceki yaklaşımlar arasındaki en önemli fark, birden çok zirveyi verimli bir şekilde yerleştiren sürüdeki bireylerin kullandığı dinamik karar alanıdır. Sürüdeki her bir birey komşularını seçmek için karar alanını kullanmaktadır ve komşularından aldığı sinyal gücüyle hareketlerini belirlemektedir. Bu biraz, bir ateş böceğinin eşlerini veya avlarını çekmek için kullandığı ışık yakıp söndürmeye neden olan lusiferine (bir enzim ile birleşerek ışın üretenbir madde) benzer olmaktadır. Daha parlak ışık daha çekici olmaktadır.

Bu algoritmanın ateş böceği algoritmasından (firefly algorithm) farkı "komşuların yeterlilik sayısı" sınırı olmaması ve mesafeye dayalı herhangi bir algı sınırı olmamasıdır (Krishnanand, 2009).

1.5.3 Karınca Koloni Optimizasyonu

Gerçek karınca koloni davranışlarının matematiksel modelleri üzerine dayalı bir algoritmadır. İlk çalışma Dorigo ve arkadaşları tarafından yapılmıştır (Dorigo, 1991). Yaptıkları çalışmada kendi sistemlerine “karınca sistemi”, elde ettikleri algoritmaya ise “karınca algoritması” adını vermişlerdir. Karınca çevre şartlarına göre besin kaynağı ile evi arasında gidebileceği yolları belirlemektedir. Belirlenen yollardan birinden ilk geçen karınca yola feremon adında bir koku bırakmaktadır. Eğer yol kısa ise bu koku daha yoğun olmaktadır ve diğer karıncalar da aynı şekilde yolda devam etmektedirler.İki yolun kesiştiği noktada karınca hangi yola gideceğini belirlemektedir. Hangi yolu seçeceğine ilk önce koku miktarının yoğunluğuna göre ikinci olarak ise gelişigüzel bir ölçüte göre karar vermektedir. Bu gelişigüzel seçimin nedeni ise bütün karıncaların aynı yolda gitmesini engelleyerek yeni ve daha kısa yolları keşfetmektir (Karaboğa, 2011).

1.5.4 Parçacık Sürü Optimizasyonu

Sezgisel yöntemlerden biri olan PSO tekniği ilk olarak kuş ve balık sürülerinin hareketlerinden esinlenerek doğrusal olmayan nümerik problemlere optimal sonuçlar bulmak için 1995–1996 yıllarında sosyolog-psikolog James Kennedy ve elektrik mühendisi Russel Eberhart tarafından ortaya atılmıştır (Kennedy, 1995). PSO popülasyon tabanlı olasılıksal bir optimizasyon yöntemi olup çok parametreli ve çok değişkenli optimizasyon problemlerine çözümler üretmek için kullanılmaktadır (Alataş, 2007).

Parçacık sürü kavramı basitleştirilmiş sosyal sistemin bir simülasyonu olarak ortaya çıkmıştır. Başlangıçtaki amaç, kuş yada balık sürü koreografisinin grafiksel olarak simülasyonlarını yapmaktı. Ancak grafiksel simülasyondan sonra, parçacık sürü modelinin bir optimizasyon yöntemi olarak kullanılabileceği keşfedilmiştir.

Kuş toplulukları gerçek yiyecek kaynağını bilmemelerine rağmen, yiyecek kaynağından ne kadar uzakta olduklarını öğrenmeye çalışırlar. Öğrenmek için izlenen yöntem yiyecek kaynağına en yakın olan kuşu izlemektir. PSO'da her bir kuş parçacık olarak, kuş topluluğu da sürü olarak temsil edilir. Parçacık hareket ettiğinde, kendi koordinatlarının uygunluk değeri yani yiyeceğe ne kadar uzaklıkta olduğu hesaplanır. Bir parçacık, koordinatlarını, hızını yani çözüm uzayındaki her boyutta ne kadar hızla ilerlediği bilgisini, şimdiye kadar elde ettiği en iyi uygunluk değerini ve bu değeri elde ettiği koordinatları hatırlamalıdır. Çözüm uzayında her boyuttaki hızının ve yönünün her seferinde nasıl değişeceği, komşularının en iyi koordinatları ve kendi kişisel en iyi koordinatlarının birleşiminden elde edilecektir (Alataş, 2007).

1.5.5 Yapay Balık Sürüsü Optimizasyonu

Yapay balık sürüsü optimizasyonu, yiyecek aramada balık sürüsünün sosyal davranışlarının benzetimi tabanlı bir zeki optimizasyon algoritmasıdır. Doğada balık, yiyeceğini besin değeri yüksek alanları tek tek arayarak ya da diğer balıklar izleyerek bulabilmektedir. Çok balıklı bölgenin besin değeri

genellikle daha yüksek olmaktadır. Bu optimizasyonun temel fikri, küresel optimuma ulaşmak için balık bireyinin yerel aramasıyla toplanma ve izleme gibi balık davranışlarını taklit etmektir. Bir yapay balığın yaşadığı ortam başlıca çözüm uzayıdır ve diğer yapay balıkların konumudur. Bir sonraki davranışı mevcut durumuna ve yerel çevresel durumuna bağlı olmaktadır. Bir yapay balık kendi faaliyetleri ve arkadaşlarının faaliyetleri yolu ile çevresini etkileyecektir (Jiang & Yuan & Cheng, 2009).

1.5.6 Bakteriyel Besin Arama Optimizasyon Algoritması

Bakteriyel Besin Arama Algoritması, E. coli bakterisinin beslenme davranışından esinlenerek karmaşık mühendislik problemlerini çözmek için geliştirilmiş bir hesaplama tekniğidir. Bakteriler, karmaşık yaşam formlarındaki diğer canlılara göre çok daha basit yapıdadırlar. Sınırlı algı ve hareket kabiliyetlerini kullanarak optimum düzeyde enerji harcayıp beslenme faaliyetlerini gerçekleştirmeleri gerekmektedir. Diğer yaşam formlarına nazaran modellenebilmeleri daha kolaydır. Bu türden canlılardan biri olan E. coli bakterisi, yapısı ve çalışma sekli en iyi anlaşılan mikroorganizmalardan birisidir. E. coli bakterisi besin maddesine ulaştığında diğer bakterileri uyarıcı etkiye sahip kimyasal bir madde salgılamaktadır. Bu madde, diğer E. Coli bakterilerinin besini bulan bakterinin bulunduğu yere doğru hareket etmesini sağlamaktadır. Eğer gıda yoğunluğu çok fazla ise bakteriler kenetlenerek grup halinde hareket edebilmektedirler (Başbuğ, 2008).

1.5.7 Kurt Kolonisi Algoritması

Bu algoritma, kurt kolonisinin sıkı bir organize sisteme sahip olmasından esinlenilerek geliştirilmiştir. Kurtlar görevleridiğerleriyle bölüşmektedirler ve avlandıkları zaman tutarlı adımlar atmaktadırlar. Az miktarda yapay kurt aktif olduğu av aralığında aramaya atanmaktadır. Arama kurtları avı keşfettiği zaman, avın konumunu diğer kurtlara ulumayla bildirmektedir.Diğer yapay kurtlar ava yaklaşmakta ve avı kuşatmaktadırlar. Kurt kolonisinin atanma kuralı, yiyeceğin ilk olarak güçlü kurda atanması ve daha sonra zayıf olana

atanmasıdır. Kurt koloni algoritması bu davranışların taklit edilmesiyle geliştirilmiştir (Liu & Yan & Wu, 2011).

1.6 Minimaks Yöntemi

Kombinatör problemlerde oyun ağacının tamamının araştırılması büyük zorluklar oluşturduğundan ağacın yalnız bir kısmına bakılmaktadır. Oyun ağacı, belli bir derinliğe kadar araştırılır, daha sonra her ara hedef durumu için özel sezgisel fonksiyon değerleri hesaplanır. Bu terminal düğümlerin değerlerinden faydalanarak köke doğru hareketlenilir ve ağacın düğümlerinin değerleri kesinleştirilir. Son olarak ise, program bu değerlere göre en iyi hamleyi gerçekleştirir. Araştırma derinliği arttıkça hesapsal olan kararlar daha çok "akıllılık" gösterecektir (Abe, 1994).

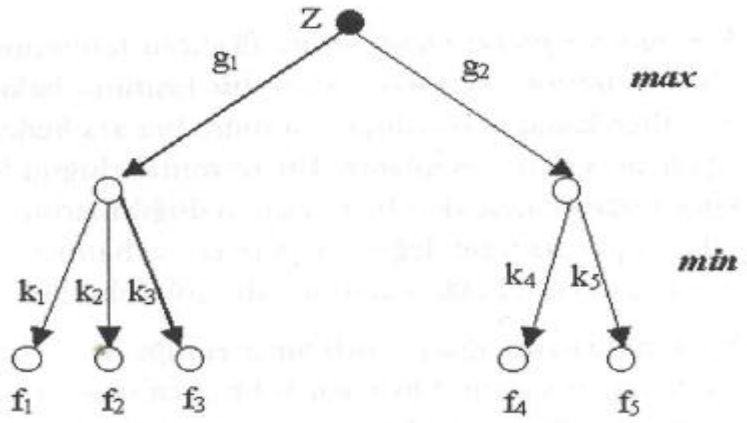
Yöntemde oyun ağacı belirli bir derinliğe kadar büyütülerek uç düğümlerin değerlendirilmesi yapılır. Önceden belirlenen değer fonksiyonu sezgisellik taşıdığından, terminal düğümlere ulaşana kadar rastlanan ara düğümlerin değerlendirilmesi çok da sağlam olmamaktadır. Yöntemde gereken derinliğe indikten sonra daha perspektifli olan gidiş yapılır. Ancak problemin karmaşıklığı göz önüne alınırsa bu derinliğin sınırlandırılması gerekmektedir. Stratejik oyunlarda taraflar, karşılıklı olarak birbirlerine üstünlük sağlamayı amaçlamaktadırlar. Taraflardan biri (biz) sezgisel fonksiyon değerini maksimuma çekmeye çalışırsa, rakip bunun tersine, yani bu fonksiyon değerini düşürmeye çalışacaktır. Böylelikle, rakip karşı tarafın (bizim) durumunu zayıflatacak en küçük fonksiyon değerini seçecektir. Buna karşılık olarak ise biz uygun seviye içerisinden en iyi gidişimizi uç durumların maksimumunu değerlendirmekle yapacağız. Görüldüğü gibi burada değer fonksiyonu büyük önem taşımaktadır. Fonksiyon değeri büyük olduğu sürece galibiyet olasılığı artır, değer küçüldüğü taktirde ise rakibin kazanma şansı artmış oluyor. Oyunculardan biri sürekli olarak yüksek (buna MAX diyeceğiz), diğeri ise küçük (MIN) değerleri takip ettiğinden yönteme minimaks ismi veriliyor (Russell & Norvig, 2003).

Aşağıdaki örneklerle O. Morgenstem ve Von Neumann tarafından önerilmiş minimaks yöntemini daha derinden ele alalım. Şekil-2'de minimaks

yöntemi ile 2 seviyeli oyun ağacının değerlendirilmesi verilmektedir. Başlangıç durumdan g_j ve g_2 geçişleri yapılabilmektedir. Her gidişe uygun olarak (örneğin g, gidişi) karşı tarafın mümkün gidişleri (k_1, k_2, k_3) değerlendirilmektedir. Uç düğümlerin (durumların) f_j sezgisel fonksiyon değerlendirilmesi yapılarak, aşağıdan yukarıya doğru hareket edilerek, kök düğümün değeri belirlenir ve mümkün hamle seçeneklerinden hangisinin en iyi olduğu kararı verilir. İlk gidişi MAX yaparsa çift derinlikler MIN'e karşılık geldiğinden, 2 derinlikli oyun ağacında MIN uç değerler içerisinden en küçük durumu seçecektir (Abiyev, 1996). (Hatırlayalım ki her derinlik 1 yarım geçişe karşı gelmektedir). Buradan Z kök düğümün değerlendirilmesi aşağıdaki şekilde olacaktır:

$Z = \max\{\min(f_1\ f_2, f_3), \min(f_4, f_5)\}$,

Örneğin, $(f_4, f_5) < f_3 < (f_1, f_2)$ ise daha avantajlı olan g_1, gidişi yapılacaktır.

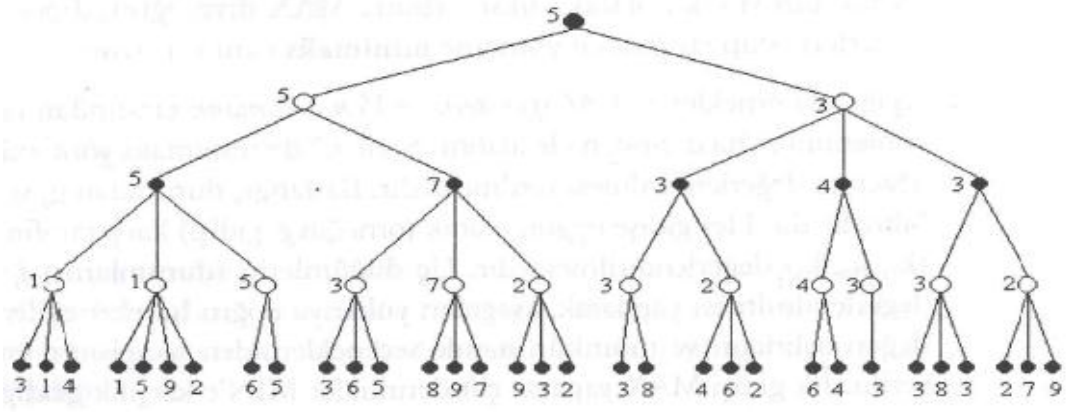


Şekil-2: Minimaks yönteminin açıklaması

Şekil-3'de 4 seviyeli bir oyun ağacı verilmiştir. Bu ağaç 2 tam gidişe (rakibin gidişleri de göz önüne alınırsa 4 yarım geçişe) karşılık gelmektedir. Burada da ardışık şekilde en alt seviyedeki fonksiyon değerleri belirlendikten sonra aşağıdan yukarıya doğru hareketlenerek kök düğümün değerleri bulunuyor ve en iyi hamle seçilir.

Kök düğüm değeri = max{min[max{min(3, 1, 4); min(l, 5, 9, 2); min(6, 5)}, {max{min(3, 6, 5), min(8, 9, 7), min(9, 3, 2)}], min[max{min(3, 8), min(4, 6, 2)}, max{min(6, 4), min(3)}, max{min(3, 8, 3), min(2, 7, 9)}]} —

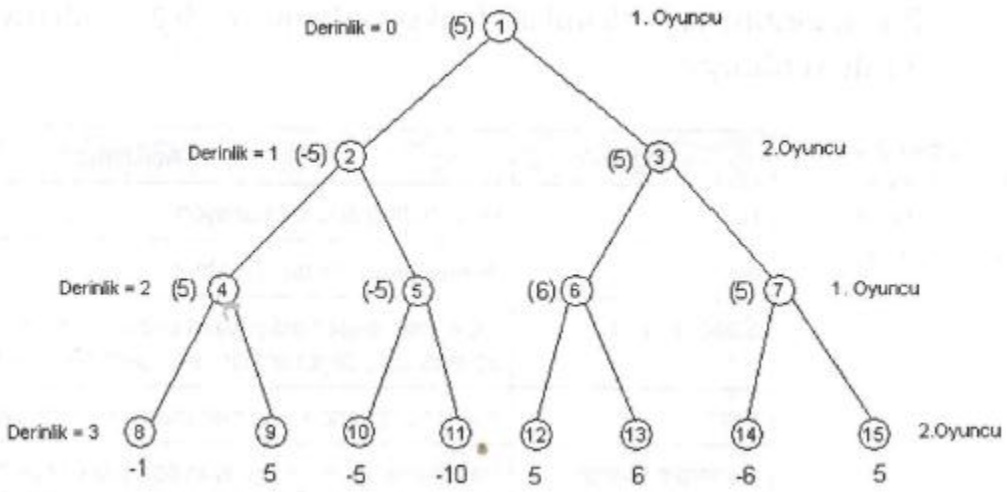
= max{min[max{l, 1, 5}, max{3, 7, 2}], min[max{3, 2}, max{4, 3}, max{3, 2,}]}= max{min[5, 7], min[3, 4, 3]} = max{5, 3,} = 5



Şekil-3: Minimaks yönteminin 4 seviyeli ağaçta uygulaması

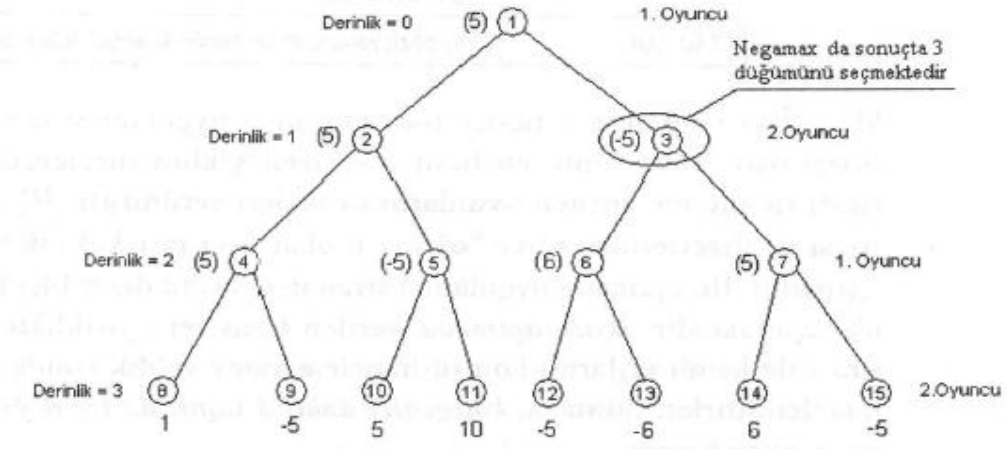
Oyun ağacı her zaman dengeli değildir. Genellikle zaman içerisinde figürleri azalan oyunda ağaç uzunlukları farklıdır. Yöntemde önemli olan, rakiplerin aynı değer fonksiyonuna sahip olmaları ve her birinin kendi açısından değerlendirmeler yapmalarıdır. Oyun ağacında MAX gidişler VEYA düğümlere, MIN ise VE düğümlere uygun gelmektedir.

Minimaks algoritması yinelemeli-derinine arama yöntemi yardımıyla gerçeklenebilir. Ancak herhangi bir seviyede maksimum seçilirken bu seviyenin altında ve üstündeki seviyelerde minimum seçilecektir. Çoğu zaman küçük bir düzenlemeyle bu durum ortadan kaldırılır: Terminal düğümlerden itibaren üst seviyelerdeki düğümlere doğru puan ataması yapılırken, seçilen minimum puanın negatifi alınarak değer atandığında, her seviye için minimum puanlı düğümün seçilmesi sağlanabilir. Bu yüzden minimax algoritması **negamaks (negamax)** olarak da bilinir. Şekil-4'de minimaks algoritması uygulanan ağaçta, negamaks algoritmasının nasıl gerçekleştiği ve bu yeni değişikliğe göre yapılan değerlendirmeler Şekil-5'de verilmiştir.



Şekil -4: 3 derinlikli bir ağacın minimaks ilkesine göre değerlendirilmesi

Görüldüğü gibi her adımda daima minimum puana sahip düğüm seçildiğinden 1. oyuncu yine 3 numaralı düğümü tercih edecektir.



Şekil-5: Şekil-4'teki ağacın negamaks ilkesine göre değerlendirmesi

Negamaks yönteminin algoritması aşağıda verilmiştir.

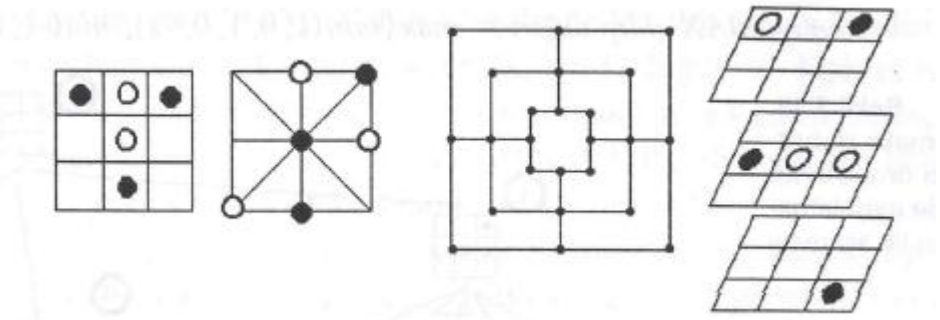
```
int NegaMax (pos, depth)
{
        if (depth = = 0) return Static_Evaluate(pos):
        best= -INFINITY;
        Child_List=Generate_Childs (pos);
        While (not Empty(Child_List))
        {   pos = Remove_One(Child_List);
            value = -NegaMax(pos, depth-1);
            if (value>best) best = value;
        }
        return best;                    }
```

Bu algoritmada kullanılan fonksiyonların ve değişkenlerin açıklanması Tablo-1'de ki verilmiştir.

Tablo-1: Negamaks algoritma değişkenleri

Değer	Açıklama
pos	Oyunda herhangi bir pozisyon
depth	Arama yapılacak derinlik sınırı
Static_Evaluate	Parametre olarak aldığı oyun pozisyonunu aktif oyuncunun bakış açısına göre değerlendirir ve bir skor geri döndürür.
Best	Herhangi bir ana kadar rastlanan en iyi pozisyon
Generate_Childs	Parametre olarak aldığı oyun pozisyonundan bir sonraki adımda erişilebilecek durumları belirler ve ilgili durum listesini geri döndürür.
Remove_One	Parametre olarak aldığı durum listesinin ilk elemanını geri döndürür ve bu durumu listeden siler
INFINITY	Pozitif büyük sayı
Child_List	Giriş pozisyonundan bir hamle ile erişebilecek durumların listesi

Minimaks yönteminin tic-tac-toe oyununda uygulamasına bakalım. Farklı çeşitleri olan bu oyunun en basit 2 seviyeli şeklini inceleyeceğiz. Şekil-6'da farklı tic-tac-toe cinsten oyunların örnekleri verilmiştir. ilk aşamada 2 oyuncu sırası ile, üzerlerine "•" ve "o" yazılı olan üçer taşı 3x3'lük 9 haneli alana yerleştirirler. İlk aşamada uygulanan strateji, oyunun diğer biçimleri için de genellik taşımaktadır. İkinci aşamada, verilen 6 taş yerleştirildikten sonra oyuncular sırası ile kendi taşlarını komşu hanelere yatay ve dik yönde olmak şartıyla hareketlendirirler. Sonuçta, köşegenler dâhil 3 taşını da bir doğru boyunca yerleştiren oyuncu kazanır (Erkalkan, 2010).



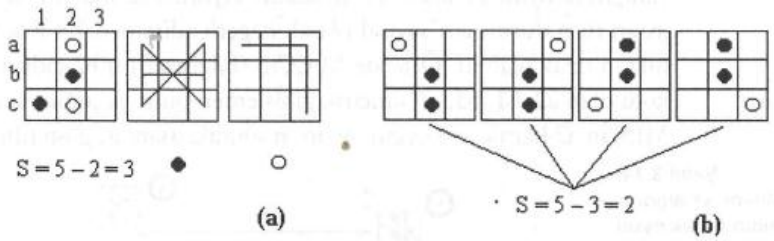
Şekil-6: Farklı tic-tac-toe oyunları

"•" taşlan ile oynayan oyuncuyu (bilgisayar) MAX, üzerinde "o" olan taşlarla oynayanı ise MIN olarak adlandıralım. Oyunun ilk aşamasında minimaks algoritmasını uygulayarak en iyi hamlelerin değerlendirilmesini gerçekleştirelim. İlk önce açıklandığı gibi durumları değerlendirmek için bir sezgisel değer fonksiyonuna gerek duyulmaktadır. Değer fonksiyonu olarak her oyuncunun, doğrular boyunca mümkün gidişler sayısı farkı göz önüne alınabilir. Yani durumlar, sezgisel S fonksiyonu yardımıyla aşağıdaki şekilde değerlendirilebilir:

S = {(MAX için yatay, dikey ve köşegen boyunca mümkün olan toplam gidiş sayısı) - (MIN için yatay, dikey ve köşegen boyunca mümkün olan toplam gidiş sayısı)}

S = ∞ oyunun kazanılması durumunda

S = -∞oyunun kaybedilmesi durumunda

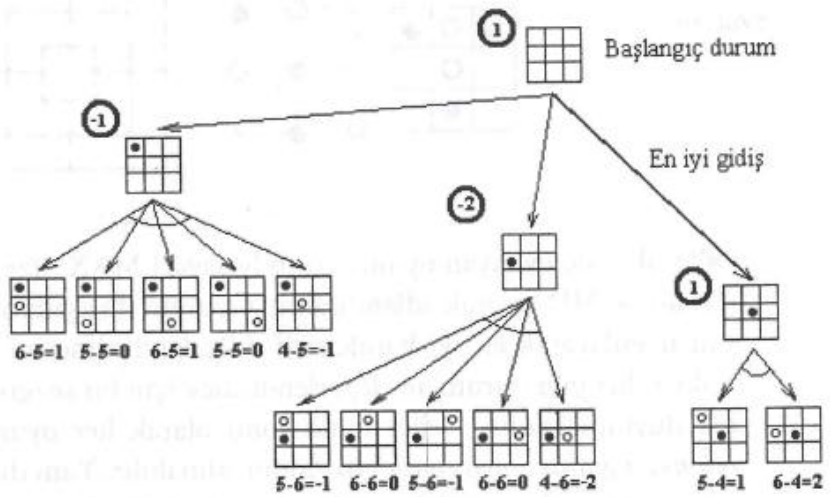


Şekil-7: Tic-tac-toe için durum değerlendirmesi(a) ve simetrik durumlar(b)

Şekil-7-a'da verilmiş örnek durum için değer fonksiyonunun hesaplanması gösterilmiştir. Simetrik durumların değer fonksiyonları eşit olduğundan, bu durumlar göz önüne alınmadığında, başlangıçtaki dallanma

sayısı önemli ölçüde küçülmüş olacaktır (Christopher, 1991). Örneğin Şekil-7-b'deki simetrik durumlar bir birinin benzeri olup sabit 2 değeri almaktadırlar.

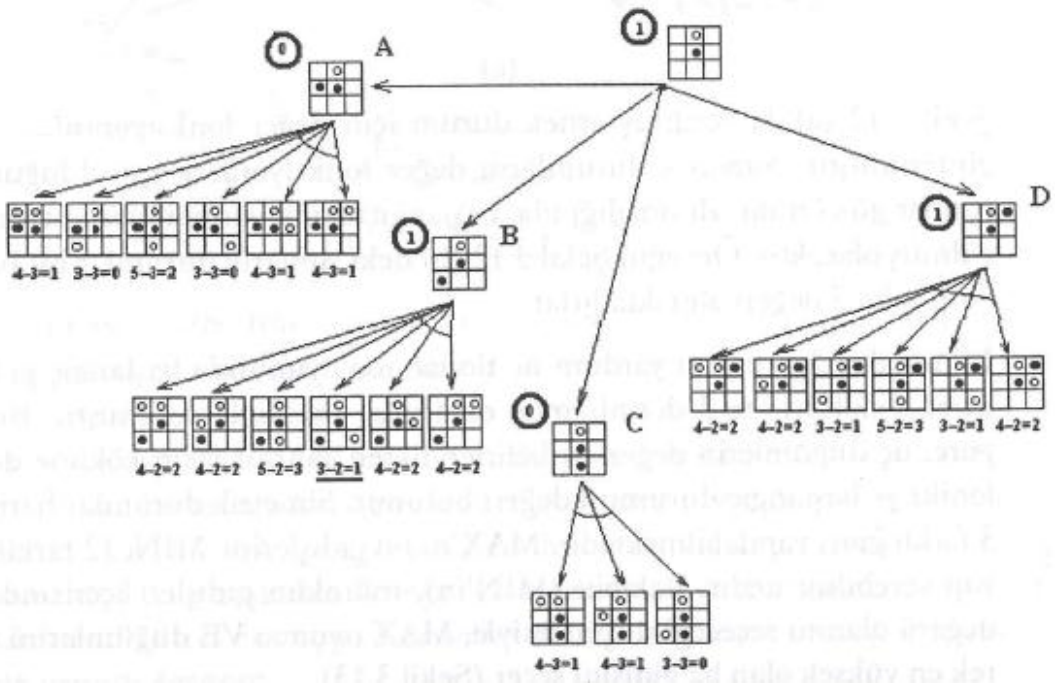
Minimaks algoritması yardımı ile tic-tac-toe oyununda başlangıç gidişin belirlenmesi, Şekil-8'da 2 derinliğinde olan ağaç üzerinde verilmiştir. Bu algoritmaya göre, uç düğümlerin değerleri belirlendikten sonra ağacın köküne doğru hareketlenilir ve başlangıç durumun değeri bulunur. Simetrik durumlar hariç, başlangıçta 3 farklı gidiş yapılabilmektedir. MAX'ın bu gidişlerine MIN, 12 farklı hamleyle cevap verebilmektedir. Rakibin (MIN'in), mümkün gidişleri içerisinden minimum değerli olanını seçeceği düşüncesiyle, MAX oyuncu VE düğümlerini değerlendirerek en yüksek olan b2 gidişini seçer (Şekil-8).En iyi MAX gidişi değeri= max{min(1, 0,1, 0, -1), min(-1, 0, -1, 0, -2), min(1, 2)}= 1.



Şekil-8: Minimaks algoritmasının örnek oyun üzerinde uygulamasının ilk aşaması

MAX en iyi gidişi olan b2'ye (aşını koyduktan sonra MIN oyuncusu kendi gidişini yapmaktadır. Eğer 2 makine tarafından karşılıklı oyun düşünülürse yine simetrik durumlar hariç, MIN mümkün al ve bl gibi iki seçeneği değerlendirerek kendisi açısından en iyi olan al gidişini yapacaktır. (Bizim açıdan MIN, kendi açısından MAX olmaktadır.) Anlaşıldığı gibi oyunda taş sınırlaması olmasa her zaman en iyi strateji seçildiğinde oyun berabere bitmektedir. Oyunlarda makine - insan karşılaşmasında ise insan tüm durumları sayısal olarak değerlendiremediğinden, onun seçimi optimumluktan uzak olabilir. Örnekte MAX'ın (bilgisayarın) b2 gidişine karşılık MIN'in daha

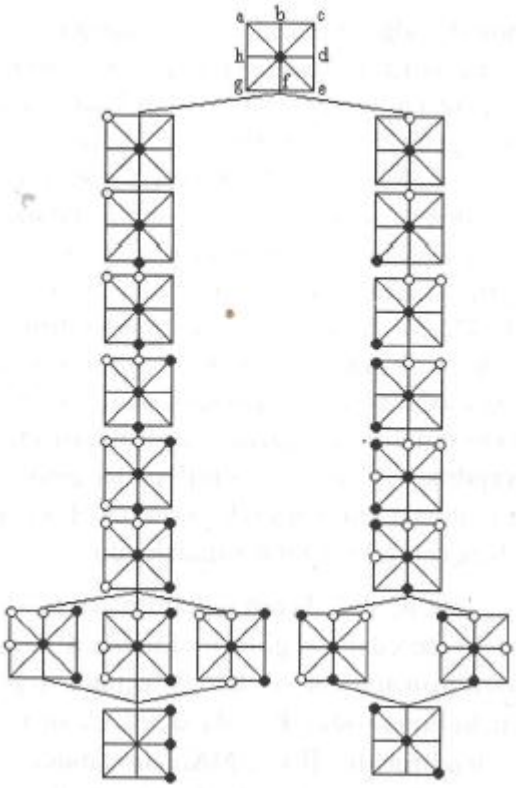
kötü olan a2, bl, b3, c2 simetrik gidişlerden birini seçtiğini kabul edelim. Şekil-9 'da MIN'in a2 hamlesine uygun oyunun sonraki aşaması gösterilmiştir.



Şekil-9: Minimaks algoritmasının örnek oyun üzerinde uygulamasının ikinci aşaması

Sonuçta aynı yönteme göre MAX, rakibin a2 gidişine karşılık olarak kendisinin en iyi olan cl veya a3 hamlesinden birini seçer (B veya D düğümü). Eğer cl hamlesi seçilirse, sıra MIN'e geldiğinde yenilgiden kurtulmak için a3 gidişinin yapılması mecburidir. Burada dikkat edilirse 2. seviyede MIN'in 2 değerli bu gidişi, MAX için c3'e nazaran daha yüksek değerli olmaktadır. Bu ise derinliğin 1 tam veya 2 yarım hamleyle sınırlı olmasından kaynaklanmaktadır. Derinlik arttıkça durum değerlendirilmesi daha sağlam olmaktadır. Eğer rakip açısından durum değerlendirilmesi yapılırsa, mümkün altı hamlenin beşi yenilgiye neden olduğundan çok büyük negatif değerler (-∞) alacaktır ve minimaks yöntemi onlar içerisinden en yüksek değerli a3 gidişini seçecektir. Sonuçta yeni bulunan durum köke koyularak, oyunun diğer aşamaları benzeri biçimde yürütülür.

İki seviyeli tic-tac-toe oyununun yukarıdaki değer fonksiyonuna göre çözüm ağacı, Şekil-10'da gösterilmiştir. Burada 1. gidişin haricinde diğer ilk 8 gidiş zorunlu olarak yapılmaktadır. Sonuçta en iyi strateji izlendiğinde ilk gidişi yapan her zaman kazanır.



Şekil-10: İki aşamalı tic-tac-toe oyununun çözüm ağacı

Görüldüğü gibi minimaks yönteminde ağacın belirlenmiş derinlikte bütün dallarının oluşturulması gerekir. Dolayısıyla "akıllıca" durum değerlendirmesi veya oluşmuş bir durumu "anlama" gerçekleştirilmemektedir. Yalnız ağacın tamamıoluşturulduktan sonra değerlendirme yapılabilir ki bu fazla zaman kaybına neden olur. Anlaşıldığı gibi, minimaksa dayalı bilgisayar programları daha çok gerçek "insani" davranışlar değil, deterministlik (algoritmik) işlevler yapmaktadır. Oyunlarda var olan gidişlerden birisinin diğerinden iyi olduğu bilindiğinde (ne derecede iyi olması önemli olmadan) hangi hamlenin yapılması gerektiğine karar verilebilir. Burada, bilgisayarlı uygulamalarda ağacın bütün dallarının değerlendirilmesi gerekmediğinden, minimaks algoritması daha etkin hale getirilebilir (Vagifoğlu, 2010)

BÖLÜM II

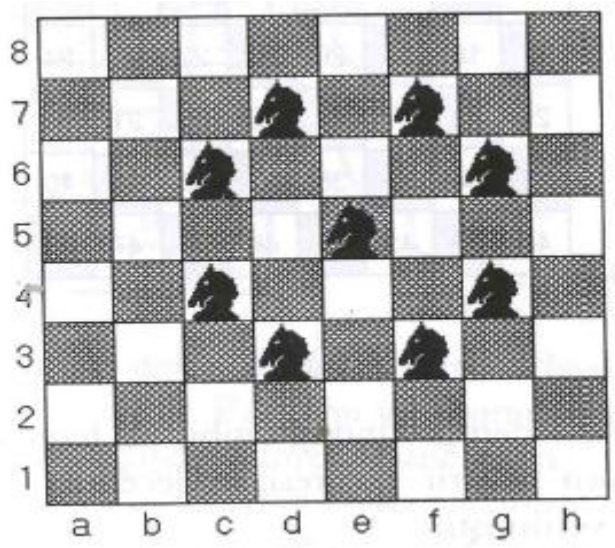
SEZGİSEL PROBLEM ÖRNEKLERİ

2.1 Sezgisel Problem Örnekleri

2.1.1 64 At Problemi

Aşağıda bakacağımız problem de satranç tahtası üzerinde atın hamlesi ile ilgilidir. Problemde, tahtanın herhangi bir hanesinden başlayarak ve her haneyi tek bir kez ziyaret etmek şartı ile at hamleleriyle tüm hanelerin gezilmesi istenmektedir [4].

Anlaşıldığı gibi bu problemde de çözüme yönelik belli bir strateji belirlenmediğinde tüm durumların taranması gerekecektir. Satranç tahtasında atın yapabileceği mümkün gidiş sayısı en çok 8'dir (Şekil-11). Atın ilerleyen hamlelerinde bu sayıyı ortalama olarak 4 kabul edersek, en iyi durumda çözümün aranması yaklaşık 463 = 8.5*1037 düğümün değerlendirilmesini gerektirecektir. (Algoritmanın üst sınırının belirlenmesinde en kötü ihtimalli değerler göz önüne alındığından, bu değer gerçekte çok daha büyüktür.)



Şekil -11: Atın mümkün gidişleri

J.C. Warnsdorf tarafından 1823 senesinde problemin çözümüne ilişkin sezgisel çözüm yöntemi önerilmiştir. Bu yaklaşıma göre her seferinde atın

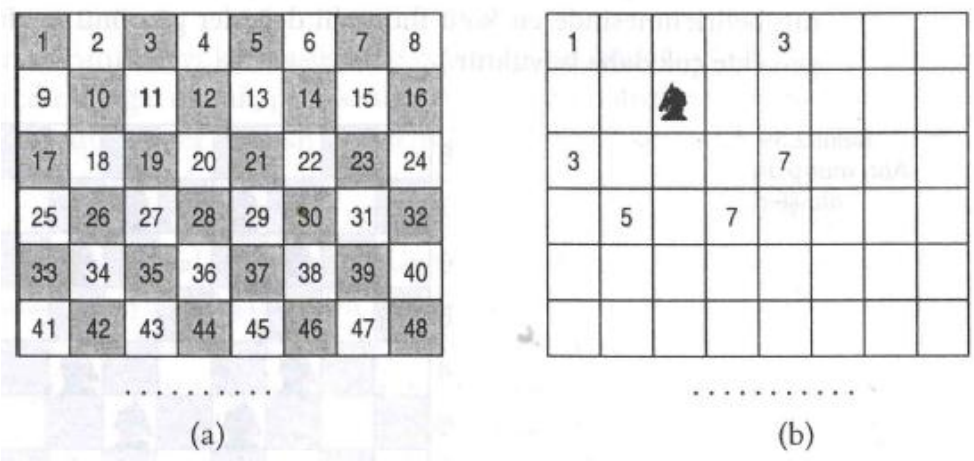
hareketi için mümkün gidişlerin gösterdiği bir sonraki haneler içerisinden en az çıkışı olan hane seçilmektedir. Dolayısıyla yöntemde, bir sonraki gidişler de göz önüne alınarak hamlelerin yapılabileceği her kare puanlandırılmakta ve sürekli olarak minimum değerler takip edilmektedir. Şekil-11'de problemin çözüm ağacı verilmiştir (Vagifoğlu, 2010).

Problemin çözümüne açıklık getirmek için haneleri sırası ile numaralandırılmış satranç tahtasını ele alalım (Şekil-12-a). Farz edelim ki, at başlangıçta 11. karede bulunmaktadır. Bu kareden onun gidebileceği haneler 1, 5, 21, 28, 26 ve 17'dir. Bu hamleler içerisinden hangisinin daha avantajlı olduğunu araştıralım.

1. haneye eriştikten sonra tek 1 kareye hareket yapılabilmektedir: 18

5'ten ---------- 3 ---------- : 15, 20, 22
21'den ---------- 7 ---------- : 4, 6, 15, 31, 38, 36, 27
28'den ---------- 7 ---------- : 13, 22, 38, 45, 43, 34, 18
26'dan ---------- 5 ---------- : 9, 20,36, 43, 41
17'den ---------- 3 ---------- : 2, 27, 34

Şekil-12-b'de problemin başlangıç gidişinin belirlenmesi için gerekli olan puanlamalar gösterilmiştir.



Şekil-12: 64 at problemi

İncelemelerden göründüğü gibi, 11. hanede olan atın bir sonraki çıkış yeri olarak minimum değerli 1 karesi seçilecektir: Ati 1-1. Problemin örnek çözümü Şekil-13 'de verilmiştir.

2	63	16	25	44	59	14	27
17	24	1	64	15	26	43	58
62	3	52	45	60	57	28	13
23	18	61	48	51	46	55	42
4	49	22	53	56	41	12	29
19	34	37	50	47	54	9	40
36	5	32	21	38	7	30	11
33	20	35	6	31	10	39	8

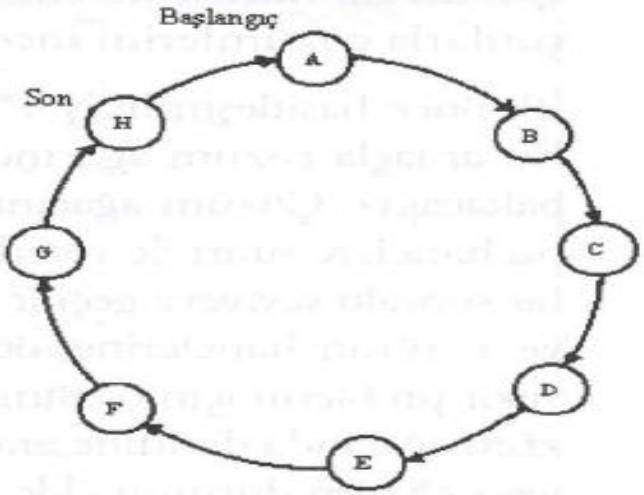
Şekil-13: 64 at probleminin örnek çözümü

Bazı kaynaklarda sezgisel olan bu yaklaşımın yanlış olarak başarı garantisinin olmamasına veya tüm haneler için çözümün bulunmadığına değinilmektedir, İlk önce problemin her zaman çözüm içerdiğini ispatlayalım. Burada yine "düğmelerve ipler" yöntemi yardımımıza gelecektir. Problemin herhangi bir çözümünün bulunduğunu varsayalım (Şekil-14). Şimdi her haneye bir düğme konulduğunu ve hanelerin at hamlelerine uygun olarak iple ilişkilendirildiğini düşünelim. Bu durumda yine kapalı çember elde edeceğiz. Çemberin kapalı olması herhangi bir düğümden çözümün mümkün olduğunu göstermektedir (Luger & Stubblefield , 1993). Örneğin Şekil-14'de verilmiş çözüm üzerinde başlangıç olarak (7, 2) hanesi istenirse (şekildeki 3 değerli hane), bu çemberi yalnızca iki değer kadar döndürmek yeterlidir (Şekil-15):

3 gidişini 1 ile; 4→2, …, 64→62; 1→63; 2→64(!)

8	23	60	41	10	25	28	43
61	40	9	24	59	42	11	26
22	7	56	63	52	27	44	29
39	62	51	6	55	58	49	12
4	21	(64)	57	50	53	30	45
35	38	5	54	(1)	48	13	16
20	3	36	33	8	15	46	31
37	34	19	2	47	32	17	14

Şekil-14: 64 at - son durum



Şekil-15: Durumların kapalılığı

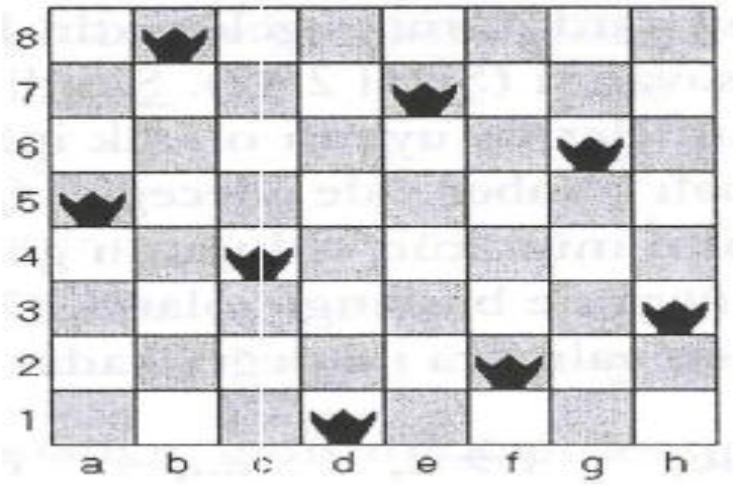
Buradan, problemin kapalılığından dolayı istenilen kareden başlandığında çözüme erişmenin mümkünlüğü görülmektedir. Fakat bu yaklaşımda, atın izlediği hareket yolu her zaman aynı olacaktır (değişen yalnızca hane numaralarıdır). Genel çözüm olarak ise bir sonraki minimum değerli haneler ele alınmaktadır. Bu durumda sezgisel değerleri aynı olan kareler oluşabileceğinden bir dallanma söz konusu olabilir. Bu dallanma sayısı ise durum uzayının bütün düğümlerini dolaşmaktan çok daha azdır ve sezgisel değerin gücü ile açıklanmaktadır. Problemdeki çözümün doğrusal karakterli olduğu görülmektedir. Bu ise belirlenmiş sezgisel fonksiyonun ne derecede güçlü olduğunu belirtmektedir. Dolayısıyla sezgisel yaklaşımla problemin çözümü her zaman bulunabilmektedir (Norvig, 1992).

2.1.2 8 Vezir Problemi

8 vezir olarak bilinen aşağıdaki problem en çok araştırılan problemlerdendir. Genel olarak problemde, n *n satranç tahtası üzerinde mümkün olduğunca en çok vezirin birbirini görmeyecek biçimde yerleştirilmesi istenmektedir.

Vezirin bulunduğu kareden yatay, dikey ve 45 derece açılar (köşegenler) boyunca hareket edebileceği düşünülürse, birbirini yemeyecek koşuluyla, n*n boyutlu tahtada en çok n vezirin yerleştirilmesi mümkündür. Standart 8*8

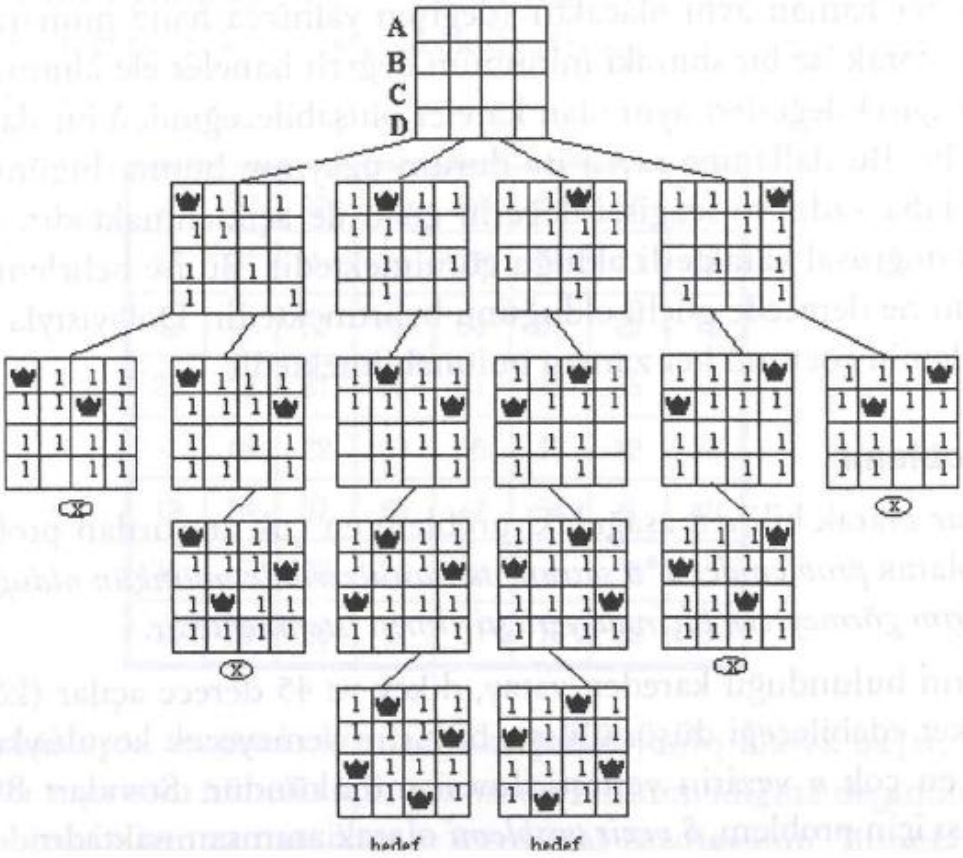
boyutlu satranç tahtası için problem, 8 vezir problemi olarak anımsanmaktadır [5].



Şekil-16:n-vezir problemi için örnek çözüm. n=8

Çözümün nasıl elde edilebileceğini araştıralım. Bu amaçla problemin farklı yaklaşımlarla çözümlerini inceleyerek sezgiselliğin önemine değineceğiz.

İlk önce basitleştirilmiş 4*4 boyutlu problemde çözümün elde edilmesini açıklayalım. Bu amaçla çözüm ağacında vezirlerin yerleştirilmesine yatay haneler boyunca sırası ile bakacağız. Çözüm ağacını oluştururken satırlar boyunca ilk boş haneye rastlandığında bu hanelere sırası ile vezirler konulur. Yerleştirilen vezirin gördüğü kareler işaretlenir ve bir sonraki seviyeye geçilir. Örneğin ilk vezir Al hanesine yerleştirilmişse 2. vezir A satırı ve 1. sütun hanelerine, öte yandan (B2, C3, D4) köşegen hanelere yerleştirilemez. 4- vezir problemi için çözüm ağacı Şekil-17'deki gibi oluşturulmaktadır. Oluşturulan çözüm ağacında derinine arama yapılır. Ağacın sol kısmından başlayarak çatışma (conflict) veya çözüm durumu elde edilene gibi ağaç aşağıya doğru büyütülür. Çatışma durumuna rastlandığında, geri dönülerek diğer dallarda çözüm aranır. Burada tek bir çözümün veya bütün çözümlerin aranması aynı ağaç üzerinde yapılmaktadır (Pamukkale Üniversitesi, 2012).



Şekil-17: 4 vezir probleminin çözüm ağacı

Problemde sezgisel bir değerlendirme yapılamadığında ağacın tüm dallarında gezinerek çözüm aranmaktadır. Durum uzayı büyüdükçe çözüm ağacının derinliği de artacaktır. Tüm dalların gezinmesi gerekliliğini düşünürsek, araştırılacak düğüm sayısı problemin boyuna bağlı olarak ri' gibi üssel bir artış gösterecektir. Bununla orantılı olarak problemin mümkün çözüm sayısı da artacaktır. Satranç tahtasının n boyuna ilişkin olarak (n=l, 2, 3,...) mümkün olan çözümler sayısı 1, 0, 0, 2, 10, 4, 40, 92,... biçiminde devam etmektedir. Fakat bu çözümlerin birçoğu (tek çözümler) diğerinden ayna yansıması ile veya simetrik dönüşümlerle elde edilmektedir. Aşağıdaki tabloda problemin farklı n boyutları için mümkün çözüm sayıları verilmiştir.

N*N problem boyu	Çözüm sayısı	Tek çözümler
1*1	1	1
2*2	0	0
3*3	0	0
4*4	2	1
5*5	10	2
6*6	4	1
7*7	40	6
8*8	92	12
9*9	352	46
10*10	724	92
11*11	2,680	341
12*12	14,200	1,787
13*13	73,712	9,233
14*14	365,596	45,752
15*15	2,279,184	285,053
16*16	14,772,512	1,846,955
17*17	95.815.104	11,977,939
18*18	666,090,624	83,263,591
19*19	4,968,057,848	621,012,754
20*20	39,029,188,884	4,878,666,808
21*21	314,666,222,712	39,333,324,973
22*22	2,691,008,701,644	336,376,244,042
23*23	24,233,937,684,440	3,029,242,658,210

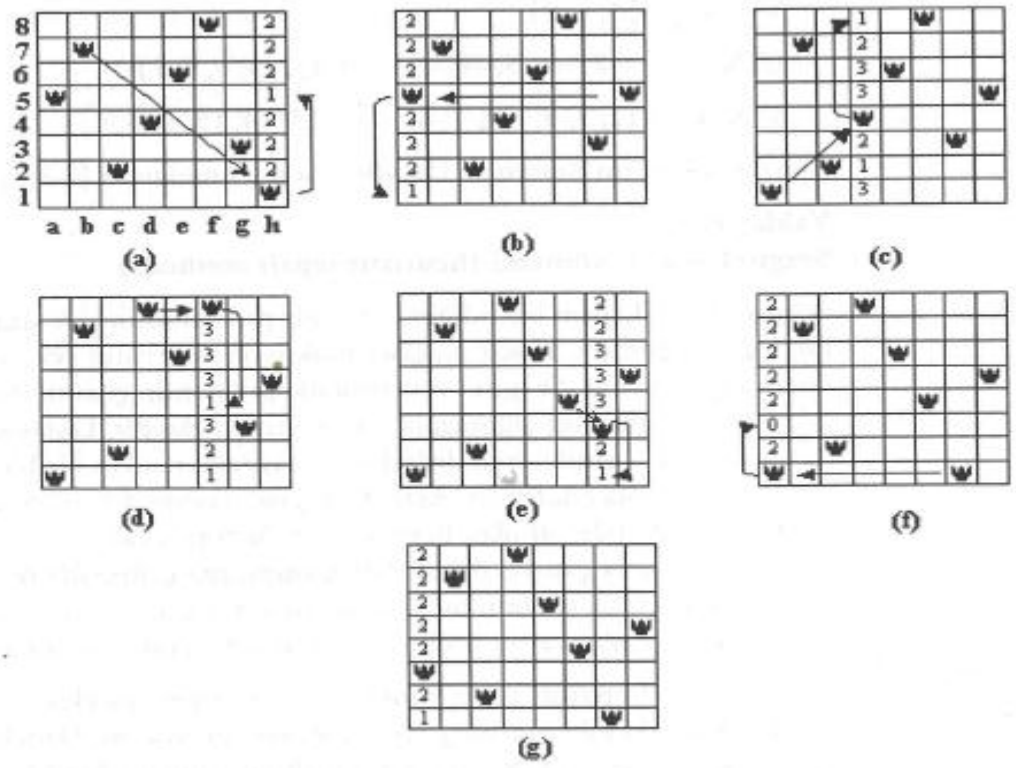
Şekil-18: Problem boyuna bağımlı olarak n- vezir probleminin çözümleri

Şekil-18'den, durum uzayı büyüdükçe çözüm sayısının da üssel olarak arttığı görülmektedir. Bu büyüme tek değerlerde daha aydın biçimdedir. Tek değerlerdeki gibi büyük sıçramalar olmasa da, çift değerlerin de kendi aralarında üssel bir artış gösterdiği anlaşılmaktadır. Aşağıda 8-vezir probleminin farklı sezgisel yaklaşımlarla çözümü incelenmektedir.

Sezgisel bir yaklaşım izlendiğinde 8 vezir probleminin çok kısa zaman içerisinde çözüldüğünü gördük. Fakat sezgisel fonksiyon değerinin iyi seçilmesi burada büyük önem taşımaktadır. Sezgisel yöntemlerde problemin çözümüne bir bütün olarak bakılmakta ve belirli bir algoritmik yol izlenmemektedir. Dolayısıyla problemi çözmede her zaman ardışık artan değerlerden faydalanmadan bir bütünün şarta göre onarımından da yola çıkılabilir. Aşağıda sezgisel onarım adı verilen yöntem incelenmektedir. Bu

yöntemde var olan herhangi bir durumda çatışan durumlar belirlenir. Sonra ise belirlenmiş çatışan durumları minimuma çekmekle problem çözümlenmeye çalışılır. Bu açıdan yöntem bir gradyan yöntemi karakteri taşımaktadır. Sezgisel onarımla, hatta 1 milyon-vezir problemi yaklaşık 50 adıma çözülmektedir (Lotfi, 1989).

Şimdi sezgisel onarım yönteminin nasıl çalıştığını inceleyelim. İlk önce vezirlerin Şekil-19-a'daki gibi rastgele yerleştirildiğini varsayalım. Örnekten de görüldüğü gibi a5'te olan vezir diğerlerinin tehdidi altında olmamaktadır. b7 hanesinde yerleştirilen 2. vezir ise hl hanesinde olan vezir tarafından tehdit edilmektedir. Çatışma oluşturan h sütunundaki hl durumu için satır değerlerini belirleyelim. Çatışan durumda olan vezirin yerleştiği sütunun eleman değerleri, her hanenin mevcut diğer vezirler tarafından tehdit edilme sayısını hesaplamakla bulunur. Şekil-19-a'daki duruma karşılık gelen h sütununun sezgisel değerler kümesi S = {*, 2, 2, 2, 1, 2, 2, 2} olmaktadır. Burada çatışma oluşturan hl hanesindeki vezir, minimum 1 ağırlıklı olan h5 hanesine çekilecektir. Şimdi a5 hanesindeki vezir çatışma durumunda oluyor, "a" sütunu için hesaplanmış yeni değerler S = {1, 2, 2, 2, *, 2, 2, 2} olacaktır (Şekil-19-b). Burada a sütunundaki vezirin yeri 1 değerli al hanesi olacaktır ve şimdi d4'teki vezir çatışan duruma düşmektedir. Bu vezir ise yalnız d2 veya d8 hanelerinden birisine çekilebilir. İşlemlerin ardışık biçimde yukarıdan aşağıya doğru yapılacağı düşünülürse bu vezirin yeni konumu d8 oluyor ve şarta uymayan f sütunundaki vezirin yerinin bulunmasına çalışılır (Şekil-19-c, d). Benzeri işlemler yapılarak bu ve sonraki vezirlerin yerleri f4 ve g1 olarak bulunur (Şekil-19-e, f). Şimdi yine a hanesindeki vezirin konumu problemin şartına uymamaktadır. Bu vezir için 0 ağırlıklı a3 hanesi belirlendikten sonra başka çatışan durumlara rastlanmadığından, sonuçta problem 6 adımda çözülmüş olur! (Şekil-19-g). Tek kelimeyle "Mükemmel" (Vagifoğlu, 2010).



Şekil-19: Sezgisel onarım yöntemi ile 8 vezir probleminin çözümü

BÖLÜM III

OYUNLARDA SEZGİSEL ÇÖZÜMLER

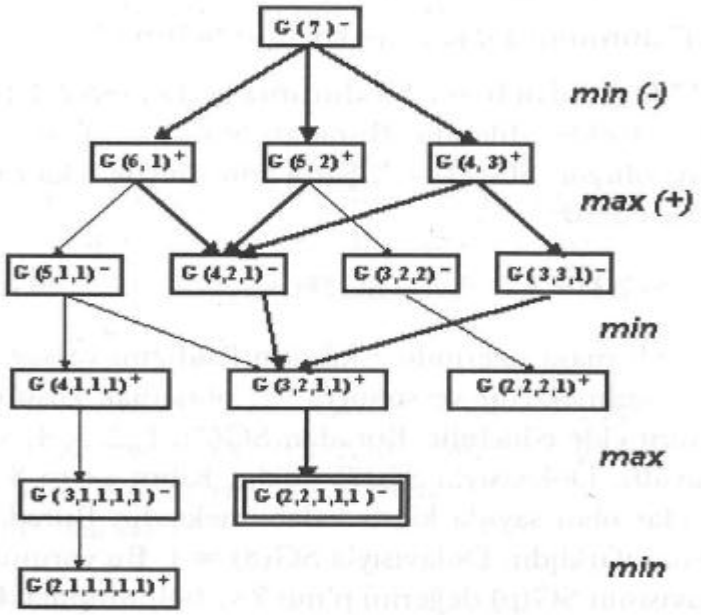
3.1 Oyunlarda Sezgisel Çözümler

Bu bölümde bilgisayarda programlanabilen sezgisel çözümlü oyun problemlerine değinilecektir. Bu problemler satranç ve dama gibi oyunlardan çok daha basit olsa da sezgisel değerlerin belirlenmesine ilişkin dikkate değer örnekler oluşturmaktadır. Oyunlar üzerinde olan bu teknikler herhangi bir problemin gerçek zamanlı uygulamasında standart olmayan farklı yaklaşımların önemini göstermektedir. Bu bölümde amaç oyunları anlatmak değildir, söz konusu oyunların bilgisayarda modellemesinde hangi sezgisel fonksiyon değerlerinin ele alındığını ve böylece en iyi stratejilerin nasıl seçildiğini göstermektir (Pohl, 1997).

3.1.1 Grundy Oyunu

Bu oyunun kuralları ise aşağıdaki şekildedir, iki oyuncu Çarşısında kibritlerin tek yığını vardır. Birinci oyuncu bu yığını 2 eşit olmayan yeni yığın şeklinde parçalar. Sonra ise her oyuncu sırası geldiğinde oluşan yığınlardan herhangi birisi üzerinde aynı işlemleri tekrarlar. Oyun, elde edilen yeni yığınlarda 1 veya 2 kibrit kalana dek devam eder. (Bu durumdan sonra oyuna devam edilmesi başlangıç şartı sağlamamaktadır.) Oyun, devam ettirilemez duruma geldiğmde gidiş sırası olan oyuncu yenilir (Bierman & Fernandez, 1993).

Farz edelim ki, yığında 7 kibrit vardır ve ilk gidişi MIN yapıyor. Notasyon olarak yığındaki kibritler sayısını ve kimin gidiş yapacağını tam sayı dizisi ve " + ", "-" işaretleri kullanarak gösterelim. Örneğin başlangıç duruma (7)¯karşılık gelecektir.



Şekil-20: Grundy oyununun ve/veya grafı

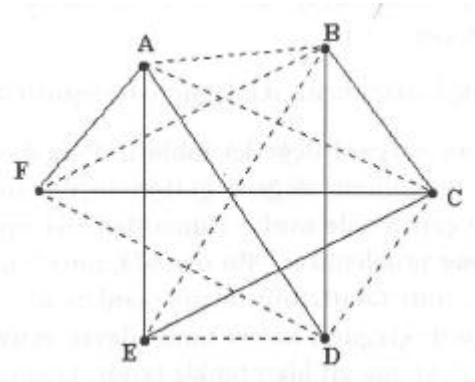
$(7)^-$ durumunda MIN oyuncunun $(6, 1)^+$,$(5, 2)^+$,$(4, 3)^+$ durumlarına gelen 3 alternatif gidişi vardır. Verilen başlangıç değerlerde oyunun sonlandırılması için 3 durum söz konusudur: $(2, 1, 1, 1, 1, 1)^+$, $(2, 2, 1, 1, 1)^{\sim}$,$(2, 2, 2, 1)^+$. Problemin şartlarına uygun durum uzayı VE/VEYA grafı biçiminde Şekil-30'dagösterilmiştir. Şekildeki kalın çizgiler çözüme uygun gelmektedir.

Şekilden de anlaşıldığı gibi MAX olan oyuncu için kazanma ancak (2, 2, 1, 1, 1) durumunda gerçekleşir. İlk başlayan oyuncunun tek sayılarda her zaman kaybedeceği ağaçtan görülmektedir (Vagifoğlu, 2010).

3.1.2 Sim Oyunları

Oyunlarda sezgisel değerlendirmelere ilişkin Sim oyununu inceleyelim. Farz edelim ki, hayali altıgenin (A, B, C, D, E, F) gibi 6 noktası verilmiştir. Her oyuncu kendi gidişinde bu noktalardan ikisini doğru ile birleştirir. Birleştirdiği doğrularla üçgen elde eden oyunculardan ilki yenilmiş olur (Vagifoğlu, 2010).

Her düğüm 5 diğer düğümle birleştirilebilir. Sonuçta 6*5=30 ilişki söz konusudur. Tekrarlanmaları göz önüne alırsak toplam yapılması gereken doğru sayısı 15 olmaktadır. Eğer oyun tamamlanmışsa ilk başlayan oyuncunun 8 doğrusunun olması gerekir. Bu doğrular 16 kez düğümlerle ilişkili olacaktır. Problemde şarta göre yalnız 6 düğüm olduğundan sonuçta bu düğümlerden en az birinden 3 doğru çıkacaktır. Sonuçta sırası gelen oyuncu bu noktaları üçgene tamamlamak zorunda kalır ve yenilir. Şekil-21'de oyunun henüz tamamlanmamış ve 14 doğrunun çizildiği durum gösterilmiştir. Görüldüğü gibi bu durumda ilk başlayan yenilecektir.



Şekil-21: Sim oyunu

Nokta sayısı arttıkça problem kambinator özellik kazanır ve sezgisel değerlendirme yapılması gerekliliği ortaya çıkar. Başlangıçta bütün doğrular eş değerli olduğundan ilk doğrunun hangi düğümleri birleştireceği önem taşımamaktadır. Birinci başlayan oyuncunun ilk gidişten sonra geriye mümkün 14 gidişi kalıyor ve her doğruya değer vererek, bir sonraki gidişlerini gerçekleştirebilir: 0 - eğer doğru, üçgene tamamlıyorsa (bu durumda o yenilir);

1 - eğer çizilen doğru, rakip hamlesini üçgene tamamlamaya zorlarsa (bu durumda rakibin yenilme ihtimali yükselir). Henüz kullanılmamış düğümleri birleştiren doğrularsa maksimum değer almış olur. Böylelikle her oyuncu kendi hamlesinde maksimum değere sahip gidişi yapıyor. Oyunun stratejisi, değerler tablosunun değerlendirilmesine dayalıdır. A. Loash tarafından önerilmiş sezgisel değerler aşağıdaki şekildedir (Herik & Uiterwijk & Rijswijck, 2002).

0: doğru, üçgeni kapatarak yenilgiye götürürse,

1: iki tarafı rakibe ait olan üçgeni tamamlayan doğru değeri,

2: tarafları farklı oyuncular tarafından oluşturulan üçgeni tamamlayan doğru değeri,

3: farklı rakiplerin doğrularını içeren 2 düğümü birleştiren doğru değeri,

4: farklı rakiplerin doğrularının olduğu düğümle, tek doğrunun çıktığı düğümü birleştiren doğru değeri.

5: bir oyuncuya ait 2 doğrunun çıktığı düğümleri birleştiren doğru değeri,

6: doğru, her iki oyuncunun yalnız bir doğrusunu içeren 2 düğümü birleştirirse,

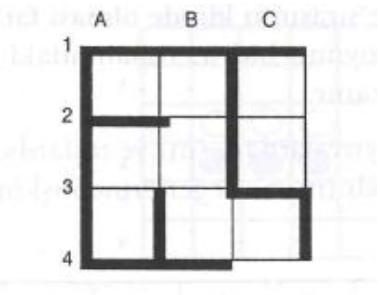
7: doğru, yalnız rakip oyuncunun doğrularını içeren 2 düğümü birleştirirse,

8: hiç kullanmamış olan düğümden geçen doğru değeri,

9: hiç kullanmamış düğümle yalnız rakibin doğrularını içeren 2 düğümü birleştiren doğru değeri,

10: hiç kullanmamış iki düğümü birleştiren doğru değeri.

Verilen sezgisel değerler tablosu daha da küçültülebilir. Fakat genel strateji, yine maksimum değerli gidişin seçilmesine dayalı olacaktır. Sim oyununun farklı çeşitleri de vardır. Bunlardan biri topoloji özellikleri göz önüne alan kare boyama problemidir (Problemin orijinal ismi "Noktalar ve kutular" (Dots and Boxes) olarak geçer). Bu oyunda, önceden boyutları belirlenmiş karenin (dikdörtgenin) taraflarının oyunculardan sırası ile çizilmesi istenmektedir. Kendi gidişinde çizgileri kareye tamamlayan oyuncu, tekrar hamle sırası onda olmakla, bu kareyi ona ait olan renkle boyar (McMillan, 1992). Oyunun örnek durumu Şekil-22'de verilmiştir.



Şekil-22: Kare boyama oyunu için örnek durum

Şekilden de görüldüğü gibi sırası gelen oyuncu, örneğin 3A yatay doğrultuda karenin tarafını çizdikten sonra kolayca, 7-2 skorlu sonuçla kazanacaktır (Vagifoğlu, 2010).

BÖLÜM IV

HAPİS TAVLASI OYUNU

4.1 Hapis Tavlası Oyunu

Tavla oyunu iki kişiyle oynanan, şans ve yetenek oyunudur. Dünyanın en yaygın yetenek oyunudur ve Roma İmparatorluğundan bu yana popülerliğini korumaktadır. Tavla oyunu Orta Doğu'da, Avrupa'da ve Yeni Dünya (Amerika'da) pek yaygındır.

Hızlı düşünme, atak davranma ve net taktikleri gerektiren bir seri oyundur tavla oyunu. Satranç ve dama oyunlarıyla karşılaştırıldığında, tavla oyunu kısa süren bir oyundur aynı zamanda ve bu nedenle genellikle bireysel oynanan oyunlardan olmayıp ciddi oyunlardan kabul edilir. Tavla oyununun nasıl oynandığını öğrenmek, basit kurallarından dolayı kolaydır ve diğer nedeni ise çok yaygındır [6].

4.2 Tavlanın Tarihi

Tavlanın tarihi binlerce yıl öncesine kadar uzanır. Milattan önce 3000 ila 1700 yıllarında tavla oyununun bir versiyonunun insanlar tarafından kullanıldığına dair kalıntılar bulunmuştur. Bir tahtanın ve pulların oluşturduğu oyun sisteminin parçaları bulunduğu düşünüldüğünde, tarihçiler, o kalıntıların

tavla oyununun ilk versiyonu olduğuna inanmaktadırlar. Bu kalıntılar muhtemelen MÖ 3000 yıllarında Eski Mısır'da yapılmış oyun araçlarıdır ve bunlara tavla yerine 'Senat' ya da 'Otuz kareli oyun' denirdi. Zarların görevi ve oyunun kuralları bilinmemektedir. Sonradan, MÖ 2600 – 1700 arasında oyun, Pers İmparatorluğu zamanında, Mezopotamya'da Sümerler zamanında yaygınlaştı.

Tavla, MÖ 1. yüzyılda başlayan, Roma İmparatorluğu zamanında da mevcudiyetini korudu, insanlar on iki çizgi anlamına gelen 'Ludus doudecim scriptorum' adlı bir oyun oynarlardı. Bu oyun, (yukarda bahsedilen) 'Senat' adlı oyunun bir çeşidi olduğu sanılıyor ve daha sonraları, günümüzün tavla kuralları na benzeyen 'Tabula' (masa oyunu) adlı oyun Senat yerine oynanmaya başlıyor. 'Tabula' oyunun gelişimi açısından tavlanın tarihinde önemli bir değeri vardır ve bunun sebebi de parayla oynanan ilk oyun olmasıdır. Bu oyun ile kumara alışılmıştır. Buna ilaveten, tavla tarihinin kitabı ilk kez yazılmıştır. Ancak ne yazık ki, bu yazılan kitap varlığını sürdürememiştir.

O zamanlar kumar yasakları uygulanıyordu, ama MÖ. 6. yüzyılda 'Alea' (zarla oynanan kumar) adlı yeni bir oyun ortaya çıktı ve yaygınlaşmaya başladı. Fazla sürmeden, İran'da 'Nard' adlı benzer bir ortaya çıktı ve popüler oldu.

MÖ. 11. yüzyılda, tavla İngiltere'de yayıldı ve aynı zamanda dünyadaki daha pek çok ülkede yaygınlaşmaya başladı; bunlardan bazıları: Almanya, Çek Cumhuriyeti, Fransa, İtalya, Çin, Türkiye, Yunanistan ve İspanya'dır. Tavla isimleri yayıldıkları ülkelerin kültürlerine göre farklı oldu. Oyunun adı olarak, Tavla terimi 1645 yılında bir İngiliz tarafından üretildi. Tavla terimi tahminen Sakson dilinden türetilmiştir ve araştırmacılar 'geri oyun' anlamında olduğunu ileri sürmektedirler.

Tavlanın Asya ve Avrupa'da yayılması uzun sürmedi ve bugün coğrafi anlamda şimdiye kadar yaygın olarak oynanan oyunlar arasında tavlanın en ünlü oyun olduğunu sanıyoruz. Tarih boyunca tavla türü oyunların pek çok çeşidi vardır ve hala benzer bir çok oyun türetilmektedir, ayrıca bu oyun türü için bir çok sayıda kitap yazılabilir [6].

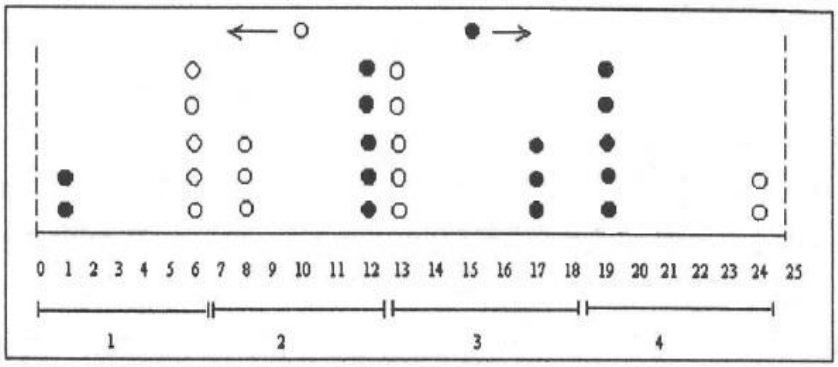
4.3 Hapis Tavlası Nedir?

Tavla pulları ve zarlar, dikdörtgenler prizması biçiminde yassı bir kutu olan tavla tahtası içinde bulunur. Oyuna başlanacağı zaman oyuncular karşılıklı oturur. Eşit büyüklükte iki kapaktan oluşan kutu açılıp kapakları birleştiren ve eşik adı verilen menteşeli kenarın bir ucu bir oyuncunun, öbür ucu rakibinin önüne gelecek biçimde ortaya konur. Kapakların oyuncuların önlerine gelen kısa kenarları 6'şar haneye bölünmüştür. Oyunun başında, oyuncular aldıkları 15'er pulu belirli bir düzenle tahtaya dizerler ve sırayla birer kere zar atarak oyuna başlarlar; atılan zarda gelen sayı pulların kaç hane ilerleyeceğini gösterir. Oyuncular attıkları zara göre hareket ettirdikleri pullarını kendi toplanma alanlarına taşırlar. Bütün pullar taşındıktan sonra gene atılan zara göre pullar toplanır. Pullarını rakibinden önce toplayıp bitiren oyuncu oyunu kazanır. Toplanma alanları tavla tahtasının bir kapağında karşılıklı olarak yer alır ve bu alanlardaki altı hane, tahtanın dış kenarlarından içeri doğru numaralanır. İki tarafın pulları toplanma alanlarına taşınırken birbirine ters yönde hareket ettirilir. Eğer bu sırada bir pul rakibin bir pulu üzerine gelirse o pulu "hapis" etmiş olur ve hapis edilen pul hareket edemez. Bir pulun hapis edilebilmesi için, o pul bulunduğu hanede tek olmalıdır. Bu durumdaki pullara "açık" denir. Hapis olan pul üzerinde ki tüm rakip pulları kalkana kadar hapis edilen pul hareket ettirilemez.

Bir oyuncunun en az iki taşının bir hanede üst üste olması durumuna "kapı" denir ve bu durumdaki taşlar kınlamaz. Kapı bulunan bir haneye rakip taş gelemez. Bu nedenle oyuncular taşlarını kapılar oluşturacak biçimde hareket ettirmeye çalışır. Oyuncular attıkları zardaki sayıya göre, istedikleri taşı oynarlar (hareket ettirirler). Eğer iki zarda da aynı sayı gelmişse bu sayı dört kere oynanır. "Gele" denen bu durumda, gele atmış olan oyuncu attığı zan oynayamaz ve zar atma sırasının yeniden kendine gelmesini bekler. Rakibi pullarını toplamaya başlamadan önce pullarını toplayıp bitiren oyuncu, rakibini "mars etmiş" olur ve iki sayı kazanır. Tavlada atılan zarların sayılan genellikle Farsça söylenir. Birden altıya kadar olan sayılara yek, dü, se, cihar, penç ve şeş denir. Çift sayılar hepyek (1-1), dubara (2-2), düse (3-3), dörtcihar (4-4), dübeş (5-5) ve düşeş (6-6) olarak adlandırılır [7].

4.4 Hapis Tavlasının Durum Uzayı

Tavla şans ve beceri özelliklerini içeren bir oyundur. Bir tavla oyununun durum uzayı Şekil-23'de verilmiştir. Burada amaç bilindiği gibi beyazlar için saat yönünde 25. numaraya, siyahlar için ise bunun tersi olan 0. numaraya taşların çıkarılmasından ibarettir. Hatta basit görünebilen bu oyunun durum uzayı 105-1010 gibi mümkün gidişlerden oluşmaktadır.

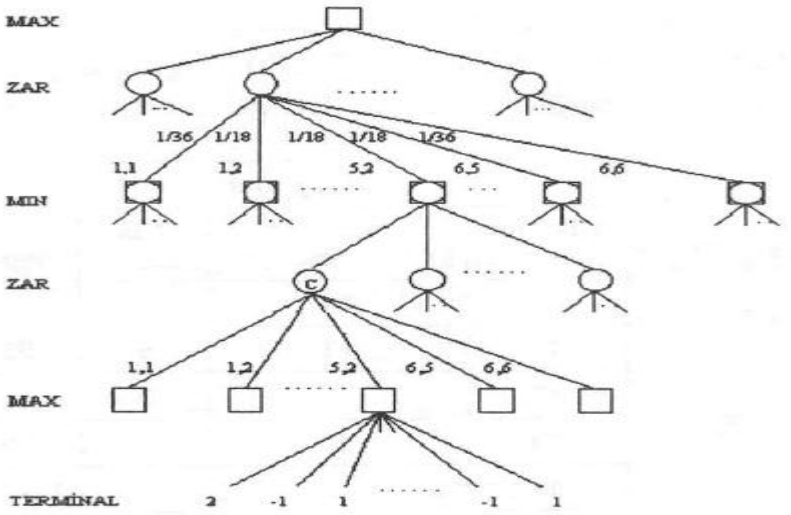


Şekil-23: Tavlanın durum uzayı

Tavlada satranç, dama veya tic-tac-toe gibi stratejik oyunlardan farklı olarak beyazlar, gidişlerinde siyahların zannın nasıl geleceğini ve onların hangi gidişi yapacağını kestirememektedirler. Buradan anlaşıldığı gibi tavlanın oyun ağacı MAX ve MIN düğümlerinin yanı sıra şans düğümlerini de içermelidir. Oyunda eşit ihtimalli 36 zar sonucu mümkündür.. Farklı zar çiftlerindeki eşit değerler göz önünde bulundurulduğunda, örneğin 1-2 ve 2-1, mümkün durum sayısı toplam 21 olur. 6 tane olan çiftli zar durumunun (1-1,2-2,3- 3, 4-4, 5-5, 6-6) gelme olasılığı 1/36, geriye kalan 15 durumun rastlanması ihtimali ise 1/18 olmaktadır. Peki, oyunda iyi gidiş karan bilgisayar tarafından nasıl verilebilir? Zar atışından sonra en iyi gidişin seçilmesini değerlendirelim. Doğal olarak mümkün gidiş uzunluğu zarlar toplamı ile sınırlıdır. Buradan anlaşıldığı gibi, yalnız beklenen değer olarak zar durumlarının mümkün gidişlerinin ortalama değeri ele alınmalıdır. Diğer stratejik oyun türlerine benzer olarak burada da terminal durumlar, sezgisel fonksiyon yardımı ile değerlendirilmektedir. Arama ağacında ilerledikte her adımda şans düğümüne erişilmektedir. Farz edelim ki "C" şans düğümüne erişilmiştir ve di; mümkün zar sonucunu göstermektedir. P(di) ise bu zarların düşme olasılığını belirlesin.

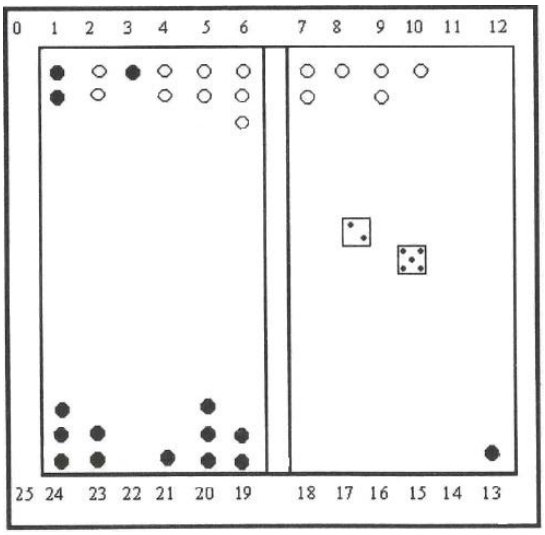
Her zar durumu için MIN'in gidiş değerlendirilmesi sezgisel fonksiyona dayalı olarak yapılır. Eğer S(C, di) olarak C'ye götüren P(di) olasılıkla mümkün zar gidişleri kümesi gösterilirse, bu düğüme (C'ye) erişmek için beklenen değer aşağıdaki biçimde hesaplanabilir (Nabiyev & Pehlivan, 2008).

$$D_{\text{Beklenen}}(C) = \sum_i P(d_i) \max_{s \in S(C,d_i)} ((s)\ şansı)$$



Şekil-24: Tavlanın oyun ağacı

Bu hesaplama sonucunda elde edilmiş değerlere göre sözü geçen minimaks algoritmasını uygulayarak MIN düğümüne erişilecektir. Kısacası benzeri oyunlarda en iyi gidişin seçilmesinde becerinin dışında bir şans da gerekmektedir. Örneğin aşağıda verilmiş oyun durumunda siyahların 2-5 atılmış zarlara göre 17 mümkün oyunu vardır. Siyahların en iyi görünen ve iki rakip taş alan (3-8),(8-10) gidişi, beyazların 4-4 zar atışı ile en kötü duruma geçebilmektedir. Bilgisayarlarda oyunların modellenmesinde programların, insan davranışlarına benzer biçimde karar vermesini sağlamak amaçlanır. Bu ise yine bir şansa gerek duyulduğunu göstermektedir ve şansın da gelme olasılığı değerlendirilebilir.



Şekil-25: Tavlada bir durum

Tavlada sezgisel fonksiyona gerek vardır. Bu fonksiyonun da biçimi polinomyal biçimdedir. Onun parametreleri ise taşların hedefe uzaklığı, kapılar, hanelerin önemi vs. şeklinde hesaplanabilmektedir (Vagifoğlu, 2010).

4.5 Hapis Tavlasının Sezgisel Yöntemler Kullanılarak Çözümü

Bilgisayarın yapacağı hamleler, mevcut durum uzayındaki taşların bulunduğu haneler tek tek zar şans faktörü ile birleşerek yeni durum uzayı elde edilir. Elde edilen her yeni uzay durumu için kullanıcının(max) bilgisayarın taşlarını hapis edebileceği hamleler derecelendirilerek belli bir olasılık elde edilir. Bu olasılıklar arasında en iyi hamleler sıralanarak zarın hamle adedi kadar oynatılır. Tıpkı yapay balık sürü algoritması ve ateş böceği sürü optimizasyonunda ki gibi.

Aşağıda bahsettiğimiz algoritmalardan esinlenilerek yazılan hapis tavlasının işleyişinin aşamaları mevcuttur.

4.5.1 Bilgisayar Hamlelerinin Genel Algoritması

Zar oynama durumu

- 1.zar (atılan birinci zarın değeri)
- 2.zar (atılan ikinci zarın değeri)
- 1.zar + 2.zar (atılan birinci zarın değeri ve ikinci zarın toplamı)
- 1.zar * 3 (çift gelen zarlarda zarın değeri x 3)
- 1.zar * 4 (çift gelen zarlarda zarın değeri x 4)

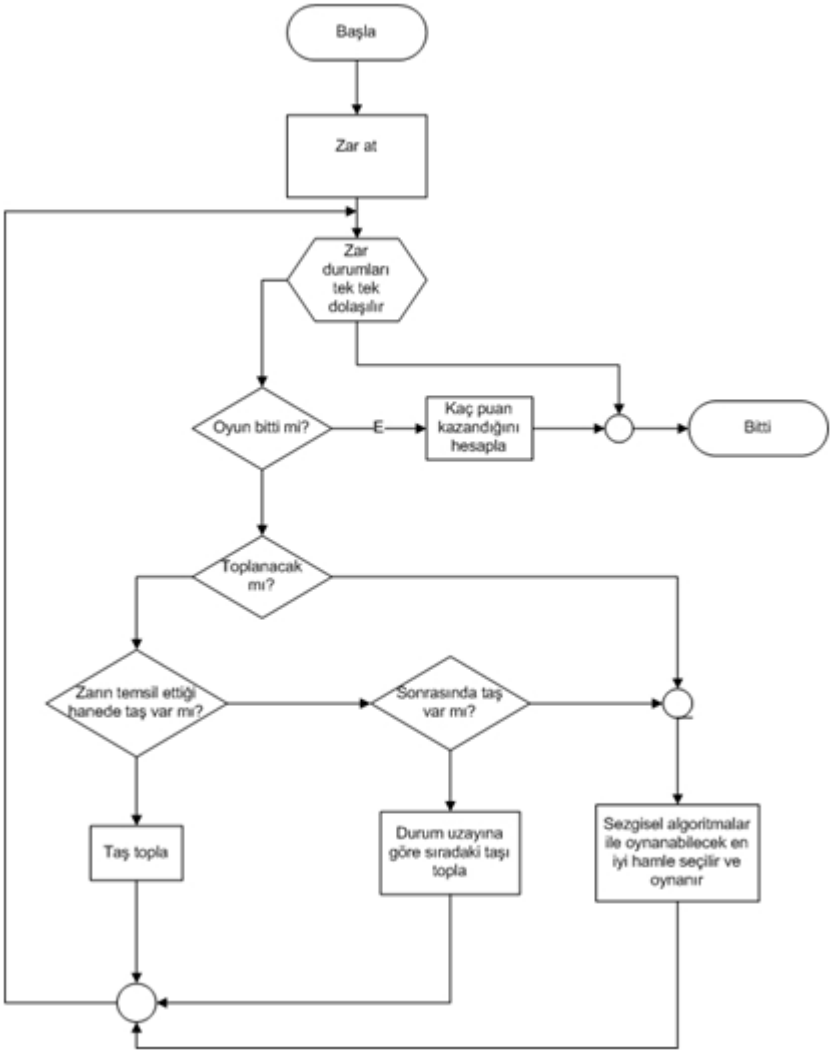
Bilgisayarın oynaması

1. Zar oynama statüleri sırayla tek tek ele alınır.
2. Bilgisayar (min) taşlarının bulunduğu haneler sırayla tek tek ele alınır.
3. Taşların gerçek konumunu geçici diziye atılır.
4. Orada ki taşı zar statüsünün değerine göre hareket ettirilir.
5. Hareket ettirilen yerde kullanıcı (max) taşı 1 den fazla varsa ya da kullanıcı (max) bilgisayar (min) taşını hapis ettiyse bilgisayar (min) bu hane için bu zar statüsünü oynayamaz. 2 ye git.
6. Kullanıcının (max) yeni oluşan durum uzayında bilgisayar (min) taşlarından kaç tanesini hapip edebiliyorsa bunlar zar ihtimallerine göre hesaplanır.
7. Hesaplanan bu değer, zar statüsü ve oynanan hane bazında oynanabilirlik dizisine atılır.
8. Seçili zar statüsü için oynanacak bilgisayar (min) hanesi varsa bir sonraki haneye git ve 2 e git.
9. Oynanacak zar statüsü varsa sıradaki zar statüsüne git ve 1 e git.
10. Oynanabilir dizisi hapis edilebilme değerlerine göre sıralanır.
11. Zarda ki hamle sayısı kadar zar statüsüne göre hapis edilme ihtimali en düşük durum uzayını oluşturan hamle yapılır.
12. Eğer tüm bilgisayar taşları toplama bölgesinde oynayabilir durumda ise toplama algoritmasını çalıştır.

Bilgisayarın toplaması

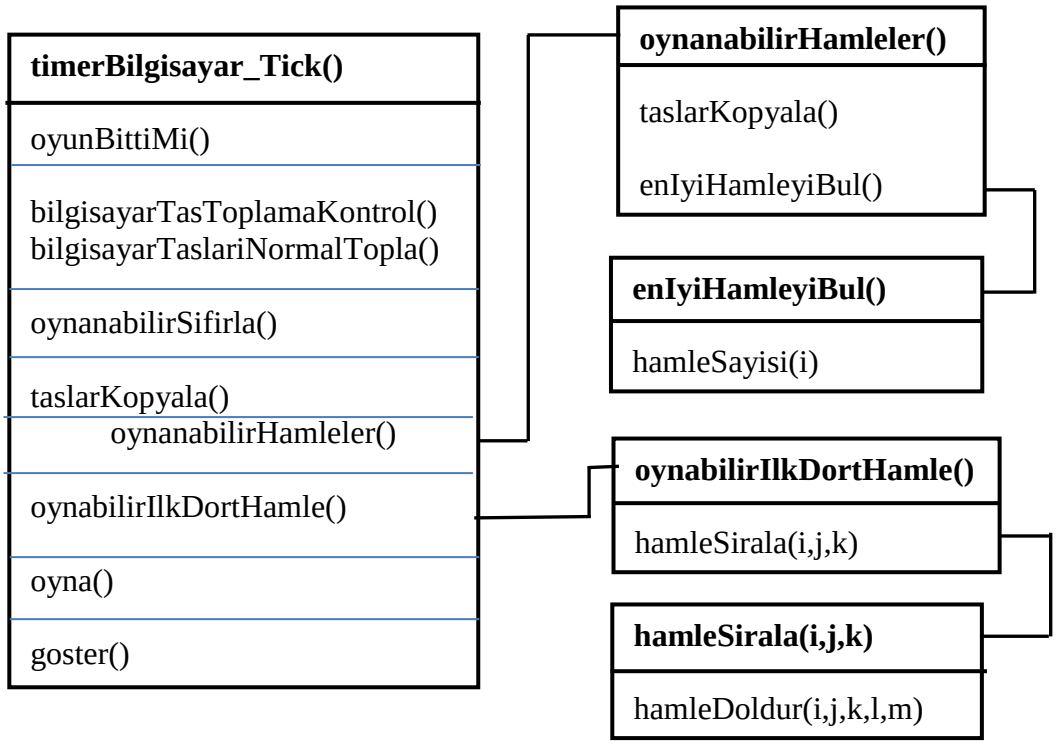
1. Oynanabilir zar değerleri sırayla tek tek ele alınır.
2. Zar değerinin bulunduğu hanede taş varsa o taş o haneden alınır ve sıradaki zar değeri için 1 e gidilir.
3. Eğer o hanede taş yoksa ve üst hanesinde taş varsa o zar değeri için oynama algoritması çalıştırılır ve sıradaki zar değeri için 1 e gidilir.
4. Durum uzayında daha küçük bir haneden taş alınır ve sıradaki zar değeri için 1 e gidilir.
5. Kullanıcının uzay durumuna göre puanlandırma yapılarak oynanan el kazanılır.

4.5.2 Genel Akış Diyagramı



Şekil-26: Genel akış diyagramı

4.5.3 Metotlar Arasındaki İlişkiler



Şekil-27: Metotlar arasındaki ilişkiler

4.5.4 C# ile Yazılmış Temel Sezgisel Metotlar

Oynanacak hamlelerin hesaplanıp yeni uzay durumunun arayüzde gösterilmesini sağlayan metot. (Ek-1)

Oynanabilecek hamleler hesaplandıktan sonra zar durumuna göre oynanarak yeni durum uzayının elde edildiği metot. (Ek-2)

Zar oynama durumuna göre mevcut durum uzayındaki bilgisayar taşlarının oynayabileceği tüm hamleler hesaplanır. Bu hamlelerin kullanıcı taşları tarafından hapisedilebilme ihtimallerini bir diziye atan metot. (Ek-3)

Hapis edilebilecek bilgisayar taşının bulunduğu hanenin zar oynama durumuna göre derecelendirilerek kullanıcı tarafından hepis edilme ihtimalini hesaplayan metot. (Ek-4)

Bilgisayarın hamle olasılıklarından en iyi dört hamlesini hesaplayan metot. (Ek-5)

Bilgisayar taşlarının hapis edilebileceği olasılığı sıralayan metot. (Ek-6)

Zar ve hane değerine göre olasılıkları diziye atan metot. (Ek-7)

Yapılan hamlenin kullanıcı tarafından hapis edilebilme ihtimalini hesaplayan metot. (Ek-8)

Toplanacak hamlenin kotrolünü yapan metot. (Ek-9)

Zara göre toplanması gerek pulu bazı kontrollerden geçirerek gerekli hamlenin yapmasını sağlayan metot. (Ek-10)

Oyunun kim tarafından kazanıldığını ve kaç puanla kazanıldığını hesaplayan metot. (Ek-11)

Oynanabilir zar ihtimallerini ve olası hamlelerin tutulduğu diziyi sıfırlayan metot. (Ek-12)

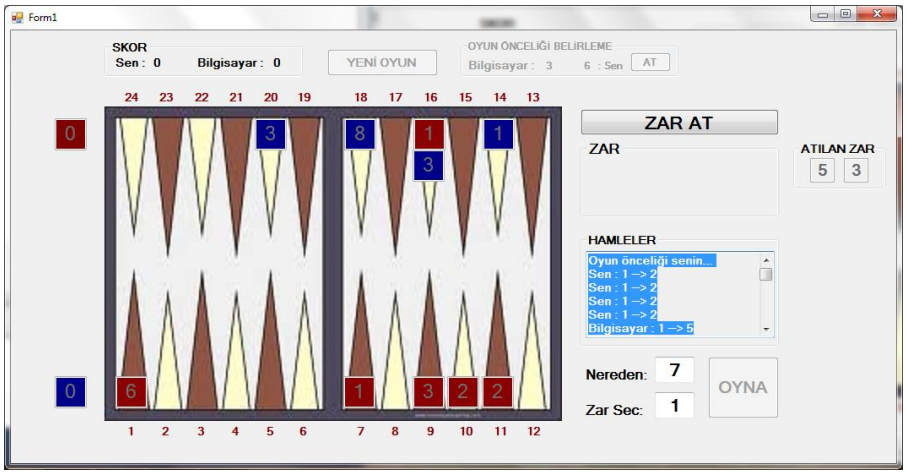
Bilgisayar taşlarının mevcut uzay durumunda en iyi hamleyi belirlemek için geçici diziye kopyalayan metot. (Ek-13)

4.6 Örnek Durum Uzayındaki Oyun Hesaplamaları ve Hamleleri

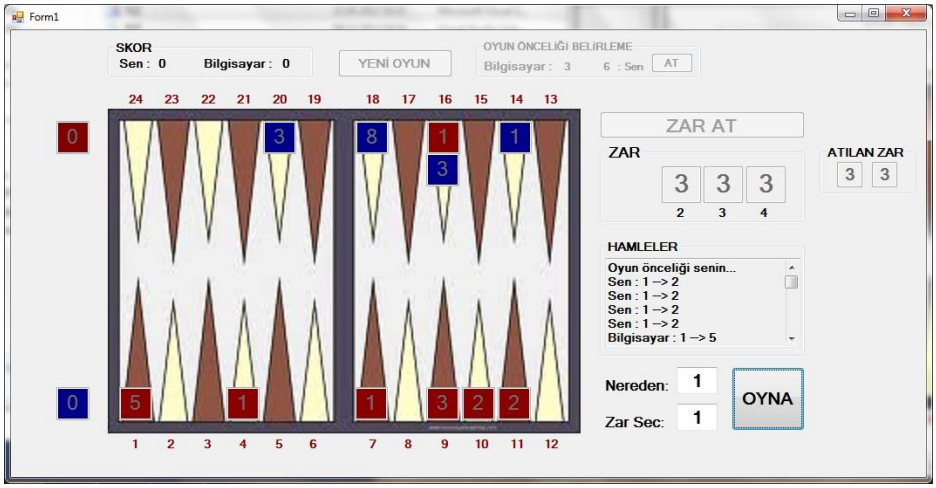
Bilgisayarın yaptığı her hamle mevcut durumuna bağlı olmaktadır. Çünkü Yapay Balık Sürüsü Algoritmasındaki gibi bir hamle kendi faaliyetlerini ve arkadaşlarının faaliyetlerinin yolu ile çevresini etkileyecektir. Ayrıca Ateş Böceği Sürü Optimizasyonunda ki gibi durum uzayındaki her bir taş komşularını seçmek için karar alanını kullanmaktadır ve komşularından aldığı sinyal gücüyle hareketlerini belirlemektedir. Çünkü bu biraz, bir ateş böceğinin eşlerini veya avlarını çekmek için kullandığı ışık yakıp söndürmeye neden olan lusiferine (bir enzim ile birleşerek ışın üreten bir madde) benzer olmaktadır. Daha parlak ışık daha çekici olmaktadır. Yani hapis edilme ihtimali yüksek olan zar parlak taş olacaktır. Amaç en çok hapis edilme riski olan taşın hapis edilme riskini azaltmaktır.

Oynama sırası kullanıcıda olan Şekil-28 de ki uzay durumunda atılan zar 3 X 3 tür.

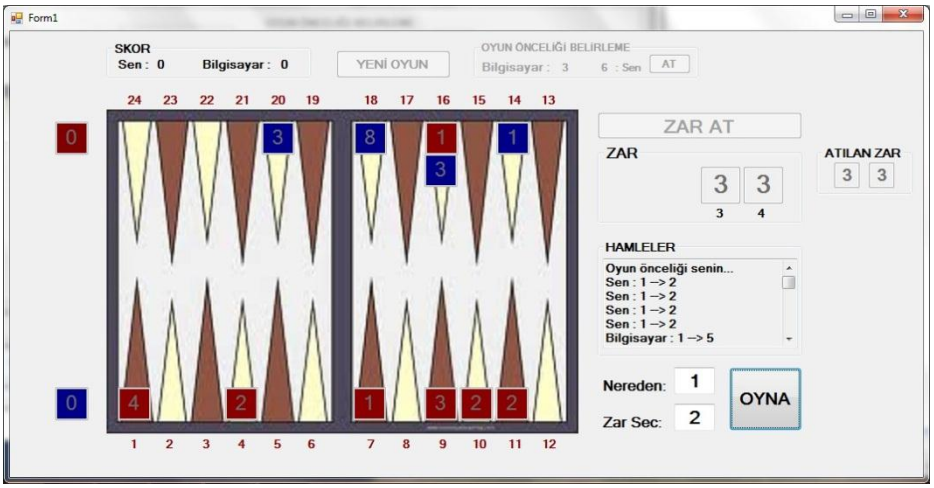
Şekil-28, Şekil-29, Şekil-30, Şekil-31'deki durum uzaylarından sırası ile kullanıcı 1. Haneden 1., 2., 3. ve 4. zarları oynamaktadır. Bu tercih tamamen kullanıcıya kalmıştır. Tabi kullanıcının yaptığı hamlelerin oynanabilirliği hamlenin tavla sınırlarının dışına çıkıp çıkmadığı, hamlenin yapılacağı yerde bilgisayar taşının 1 den fazla olup olmadığı ya da kullanıcı taşının hapis edilip edilmediği ile kontrol edilir. Bu kontroller sonrasında kullanıcının oynamak istediği hamle gerçekleştirilir.



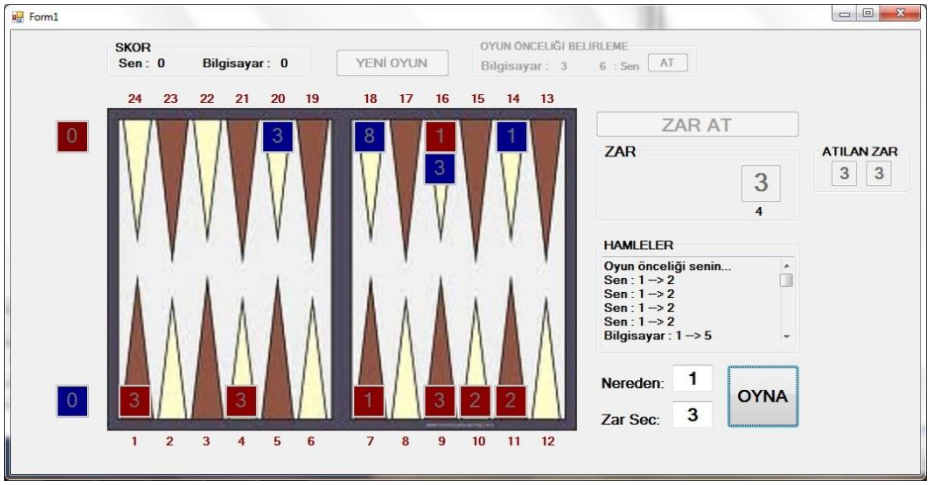
Şekil-28: 1. Durum uzayı



Şekil-29: 2. Durum uzayı



Şekil-30: 3. Durum uzayı



Şekil-31: 4. Durum uzayı

Sıra bilgisayara geldiğinde ilk başta otomatik olarak zar atılır. Sonra atılan zarın oynayabileceği hamleler ve bu hamleler sonucunda doğacak uzay durumları tek tek ele alınır. Her uzay durumu için kullanıcının atabileceği zar ihtimalleri ile çift zar, tek zar ve iki zarın toplam değerleri derecesinde bilgisayar taşlarını hapis edebilecekleri oranlar hesaplanır. Bu hesapta ortaya çıkan en düşük değer bilgisayarın oynayacağı hamleyi belirler. Şimdi Şekil-32'de ki durum uzayı için bu ihtimalleri hesaplayıp en düşük değeri tespit edelim.

Kullanacağımız bazı kısaltmaların açıklamaları aşağıda ki gibidir.

tekZar : Bilgisayarın atılacak tek zar değeri ile hapis edilebileceği hamle sayısı

ikiZarınToplamı : Bilgisayarın atılacak 1. zar ve 2. Zarın değerlerinin toplamı ile hapis edilebileceği hamle sayısı

çifZar : Bilgisayarın atılacak çift zarlar değeri ile hapis edilebileceği hamle sayısı

zarToplamı : (tekZar * (1 / 6) + ikiZarınToplamı * (1 / 18) + çifZar * (1 / 36)) bu formül ile durum uzayında kullanıcının bilgisayar taşlarını hangi seviyede hapis edilebileceği hesaplanır

Atılan zar: 2 X 3

20. hanede bulunan taş için;

Değeri 2 olan 1. zar için;

tekZar : 3
ikiZarınToplamı : 16
çifZar : 7
zarToplamı : 1,583

Değeri 3 olan 2. zar için;

tekZar : 4
ikiZarınToplamı : 30
çifZar : 14
zarToplamı : 2,722

Değeri 2 ve 3 olan zarların toplamı için;

tekZar : 6
ikiZarınToplamı : 32
çifZar : 16
zarToplamı : 3.222

Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

18. hanede bulunan taş için;

Değeri 2 olan 1. zar için;

tekZar : 3

ikiZarınToplamı : 16

çifZar : 7

zarToplamı : 1,583

Değeri 3 olan 2. zar için;

tekZar : 4

ikiZarınToplamı : 32

çifZar : 16

zarToplamı : 3,222

Değeri 2 ve 3 olan zarların toplamı için;

tekZar : 7

ikiZarınToplamı : 28

çifZar : 21

zarToplamı : 3.305

Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

16. hanede bulunan taş için;

Değeri 2 olan 1. zar için;

tekZar : **0**

ikiZarınToplamı : **0**

çifZar : **0**

zarToplamı : **0**

Değeri 3 olan 2. zar için;

tekZar : 7

ikiZarınToplamı : 28

çifZar : 21

zarToplamı : 3,305

Değeri 2 ve 3 olan zarların toplamı için;

Bilgisayar bu hamleyi yapamadığı için hesaplama yapılmaz

Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

14. hanede bulunan taş için;

Değeri 2 olan 1. zar için;

tekZar : 4

ikiZarınToplamı : 12

çifZar : 8

zarToplamı : 1,555

Değeri 3 olan 2. zar için;

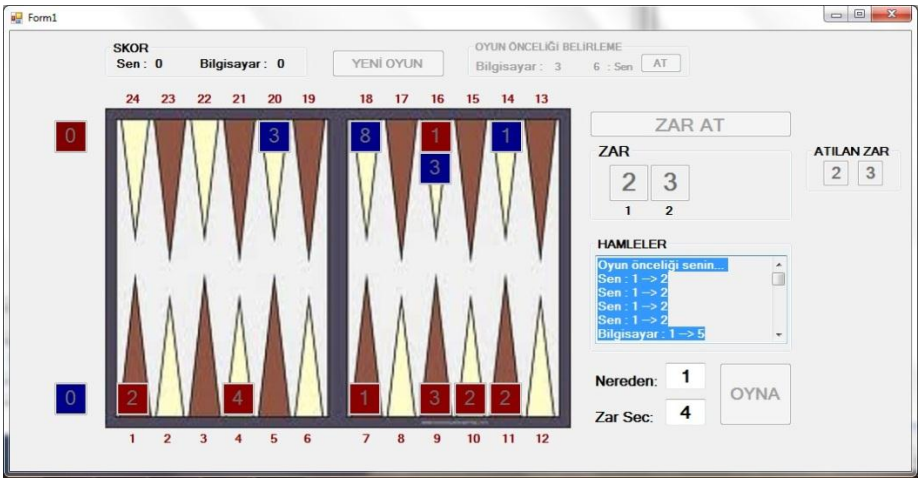
Bilgisayar bu hamleyi yapamadığı için hesaplama yapılmaz

Değeri 2 ve 3 olan zarların toplamı için;

Bilgisayar bu hamleyi yapamadığı için hesaplama yapılmaz

Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

Elde edilen bu değerler sonucunda en düşük değere sahip olan 16. hanede bulunan taş için değeri 2 olan 1. zar oynanır.



Şekil-32: 5. Durum uzayı

Şekil-33'de ki durum uzayında da görüldüğü gibi 2 değerine sahip 1. zar oynandı. Şimdi oynanacak 3 değerine sahip 2. zar. Bunun için Şekil-33'de ki durum uzayı için bu ihtimalleri hesaplayıp en düşük değeri tespit edelim.

Atılan zar: 2 X 3

<u>20. hanede bulunan taş için;</u>

Değeri 2 olan 1. zar için;

Bu zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

Değeri 3 olan 2. zar için;

tekZar : **1**

ikiZarınToplamı : **12**

çifZar : **6**

zarToplamı : **0,999**

Değeri 2 ve 3 olan zarların toplamı için;

Değeri 2 olan 1. zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

18. hanede bulunan taş için;

Değeri 2 olan 1. zar için;

Bu zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

Değeri 3 olan 2. zar için;

tekZar : 3

ikiZarınToplamı : 13

çifZar : 8

zarToplamı : 1,444

Değeri 2 ve 3 olan zarların toplamı için;

Değeri 2 olan 1. zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

16. hanede bulunan taş için;

Değeri 2 olan 1. zar için;

Bu zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

Değeri 3 olan 2. zar için;

tekZar : 4

ikiZarınToplamı : 12

çifZar : 14

zarToplamı : 1,722

Değeri 2 ve 3 olan zarların toplamı için;

Değeri 2 olan 1. zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

14. hanede bulunan taş için;

Değeri 2 olan 1. zar için;

Bu zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

Değeri 3 olan 2. zar için;

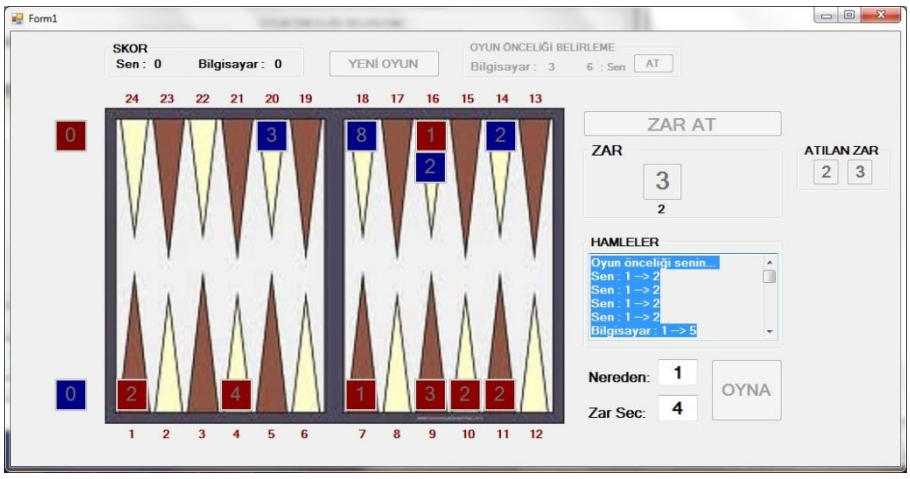
Bilgisayar bu hamleyi yapamadığı için hesaplama yapılmaz

Değeri 2 ve 3 olan zarların toplamı için;

Değeri 2 olan 1. zar bir önceki durum uzayında oynandığı için hesaplama yapılmaz.

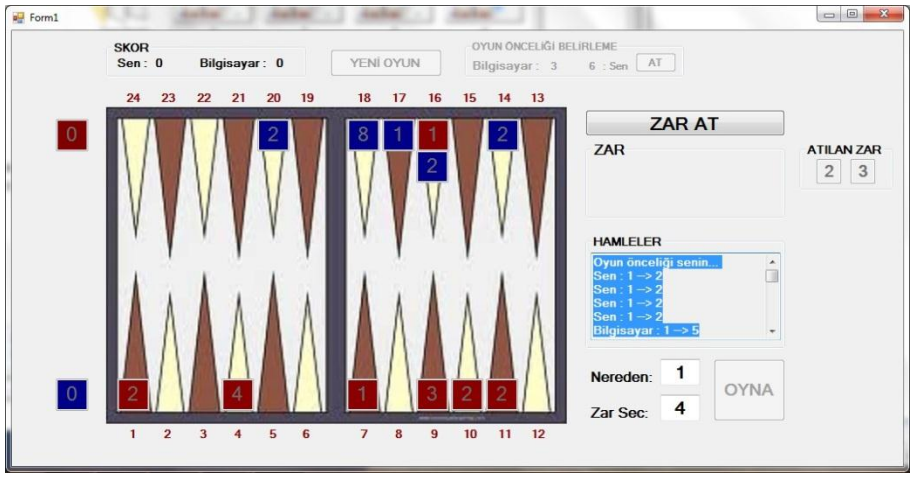
Atılan zar çift olmadığı için çift zar için hesaplama yapılmaz

Elde edilen bu değerler sonucunda en düşük değere sahip olan 20. hanede bulunan taş içindeğeri 3 olan 2. zar oynanır.



Şekil-33: 6. Durum uzayı

Şekil-34'de de bilgisayarın hamlelerini tamamladıktan sonraki durum uzayı mevcuttur.

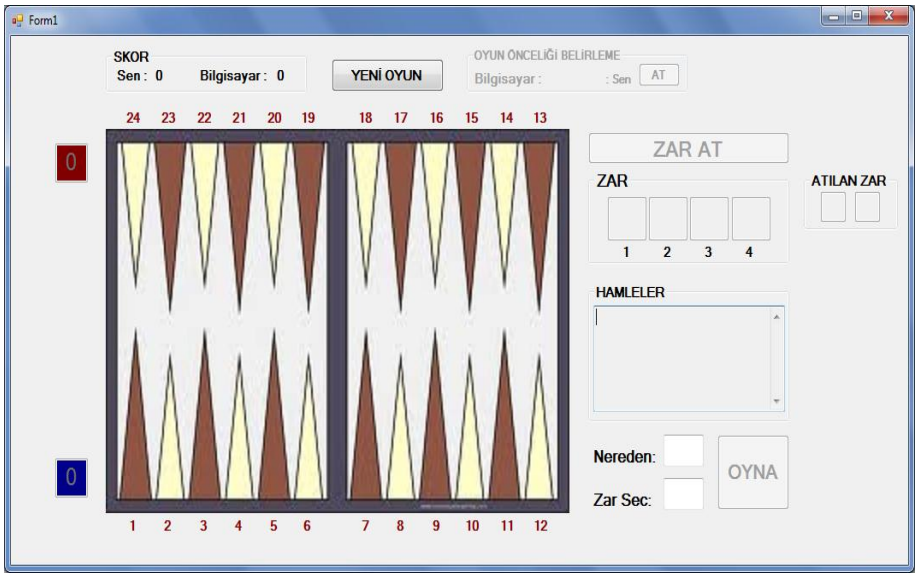


Şekil-34: 7. Durum uzayı

4.7 Hapis Tavlası Yazılımının İşletilmesi

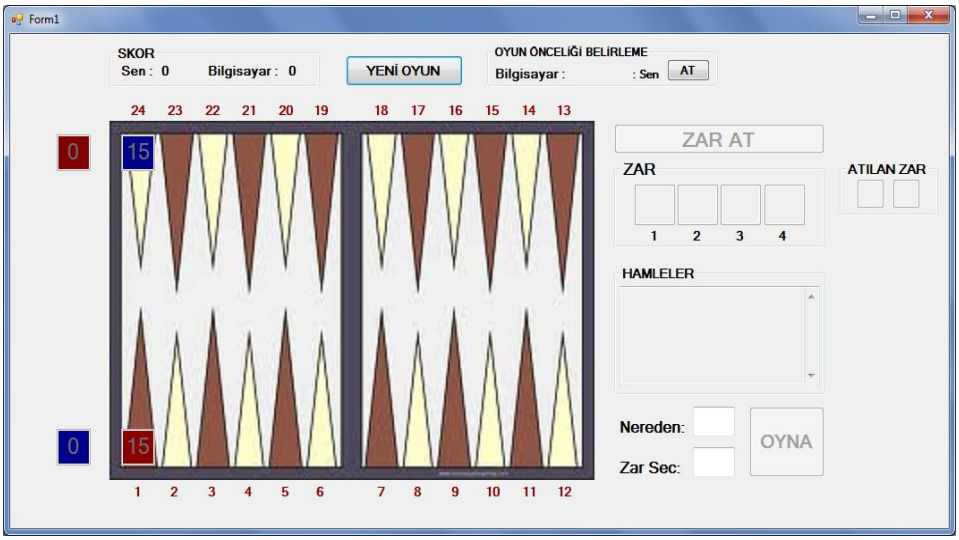
Daha önce üzerinde tez çalışması yapılmayan hapis tavlası, bu tezde sezgisel algoritmalar kullanılarak yazılmıştır. Bu çalışmanın uygulama ara yüzü ve gerekli oyun aşamaları aşağıdaki gibidir.

Şekil-35 deki "YENİ OYUN" butonu ile tüm taşlar 1'ler hanesine konumlandırılıp taşlar oyuna hazır hale getirilir.



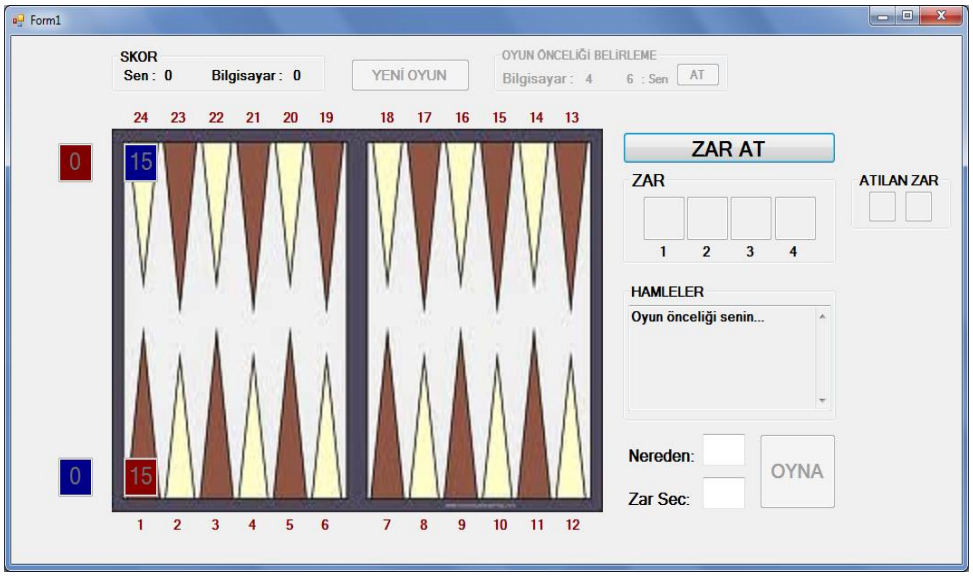
Şekil-35: Tavlada yeni oyuna başlama aşaması

Şekil-36'da görüldüğü gibi aktif hale gelen "AT" butonu ile zar atılarak yüksek atan taraf oyuna başlamaya hak kazanır.



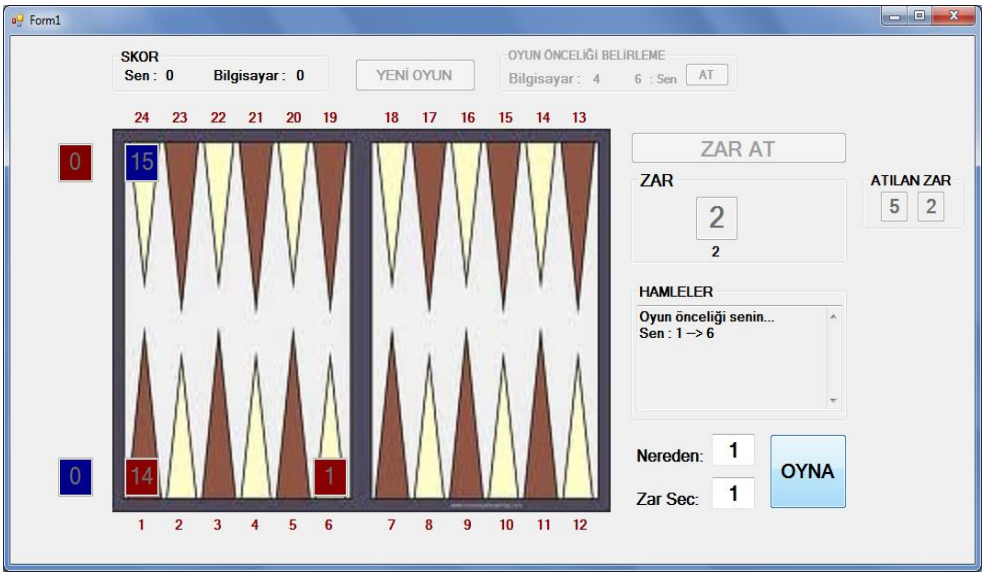
Şekil-36: Oyuna başlayacak tarafın belirlendiği aşama

Şekil-37'daki "ZAR AT" butonu ile kullancının oynayacağı zarlar belirlenerek "ZAR" gurubu adı altında zarların değerleri yazılır. Aynı zamanda atılan zarlar "ATILAN ZAR" grubu altında gösterilir.



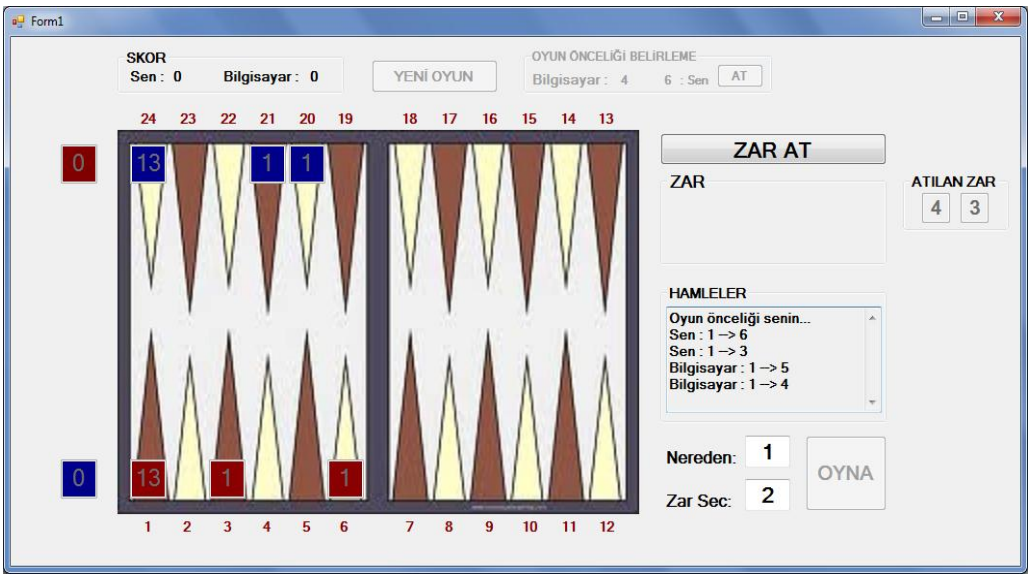
Şekil-37: Kullanıcının oynayacağı zarı belirleme aşaması

Şekil-38 aktif halde olan "Nereden" kısmına oynamak istediğimiz taşın konumunu, "Zar Seç"kısmınada oynamak istediğimiz zarın numarasını yazıyoruz. Daha sonra "OYNA" butonu ile hamle yapılır. Ayrıca yapılan her hamleler "HAMLELER" bölümünde gösterilmektedir.



Şekil-38: Kullanıcının zarı oynayacağı aşama

Şekil-39 de görüldüğü gibi kullanıcı oynayacağı son zarı oynadıktan sonra "OYNA" butonu pasif hale gelir ve bilgisayara sıra geçer. Bilgisayarda zar atıp sezgisel algoritmalar ile hamlesini yaptıktan sonra tekrar sıra kullanıcıya geçer.



Şekil-39: Kullanıcının zar için oynadığı son hamle ve sonrasında bilgisayarın oynaması aşaması

Tüm taşlar toplama hanesinde toplandıktan sonra tavlanın yanında bulunan toplama havuzunun değerini 15 ulaştıran ilk oyuncu oyunun galibi olup skor "SKOR" bölümünde gösterilir.

BÖLÜM V

SONUÇ VE ÖNERİLER

Optimizasyon problemleri için birçok algoritma önerilmiştir. Sezgisel algoritmalar, büyük boyutlu optimizasyon problemleri için, kabul edilebilir sürede optimuma yakın çözümler verebilen algoritmalardır.

Doğadaki birçok süreç, olay ya da model incelenerek değişik sürü zekâsı tabanlı yöntemler geliştirilip farklı problemler için etkili çözüm bulmak amacıyla kullanılabilir. Sezgisel algoritmaların her biri tüm problemler için en iyi sonucu vermeyebilir. Bir problem için iyi çözüm veren bir yöntem, başka bir problem için kötü olabilir. Bu yüzden algoritmaların iyi çözüm verdiği problem tipleri belirlenmelidir.

Bu tez çalışmasında uygulaması yazılan hapis tavlasında kullanılan yapay balık sürü ve ateş böceği sürü optimizasyon algoritmaları sürü zekası adı verilen sınıfta yer almaktadır. Yapılan çalışmada görülmüştür ki algoritmaların parametrelerinin doğru şekilde seçilmesi algoritmanın etkinliğini arttırmaktadır.

Hapis tavlası iki kişi ile oynanan bir oyundur. Bu çalışmada oyunculardan biri insan zekası olurken diğeri sürü zekası algoritmalarından yapay balık sürü ve ateş böceği sürü optimizasyon algoritmaları kullanılarak en iyi hamleyi seçen bilgisayardır. Yapay balık sürü algoritmalarındaki amaç, bir yapay balığın kendi faaliyetleri ve arkadaşlarının faaliyetleri ile çevresini etkilemektir. Ateş böceği sürü optimizasyon algoritmalarındaki amaç ise sürüdeki her bir birey komşularını seçmek için karar alanını kullanmaktadır ve komşularından aldığı sinyal gücüyle hareketlerini belirlemektir. Daha önce üzerinde tez çalışması yapılmamış hapis tavlası oyununda bilgisayar en iyi hamleyi kullanıcının ve durum uzayında bulunan diğer taşların etkisinde kalarak en doğru hareketi yapmaktadır. Bu şekilde sürü mantığı ile hapis tavlası oyun kuralları dahilin de hem rakibin hamlelerini önceden sezinlemektedir hem de tüm taşlarını bir birleri ile etkileşim içinde tutmaktadır.

Yapay balık sürü ve ateş böceği sürü optimizasyon algoritmaları mantığına dayalı yazılan bu hapis tavlası uygulaması ile 4 kez test yapılmıştır.

İlk elde zarı yüksek atan kullanıcı oyuna başlamıştır. Bilgisayar güzel hamleler ile kullanıcıya zor anlar yaşatsa da kullanıcı bu elde 1 puan ile galip gelmiştir. Kullanıcı - Bilgisayar: 1 - 1

İkinci elde zarı yüksek atan yine kullanıcıydı. Kullanıcının toplama alanında bilgisayar taşı hapis edilmiş durumdayken kullanıcı 3 adet taş toplamıştır. Bilgisayarın yenilmesi beklenirken hapis edilen taşın boşta kalması ile bu sefer bilgisayar kullanıcı taşını hapis etmeyi başarmıştır. Hızlı bir şekilde taşları toplama alanında biriktiren bilgisayar yüksek değerli zarlar ile durumu eşitlemiştir. Kullanıcı - Bilgisayar: 1 - 1

Üçüncü elde bu sefer zarı yüksek atan bilgisayar oyuna başlamıştır. Oyunun her anında baskın olan bilgisayar eli kazanan taraf olmuştur. Kullanıcı - Bilgisayar: 1 - 2

Dördüncü elde tekrar zarı yüksek atan bilgisayar oyuna başlamıştır. Bol hapis edilmiş taşlar ile gecen elin galibi bu sefer kullanıcı olmuştur. Kullanıcı - Bilgisayar: 2 - 2

Test sonucunda galip olmasa da seyri ve zekası güzel bir oyun ortaya çıkmıştır. Atılan zar ve yapılan hamleler ile oluşan durum uzayındaki farklı taş dağılımı oyunun seyrini değiştirmeyi başarmıştır.

Hapis tavlasını mobil ortamada taşıyıp daha cazip hale getirilebilir.

Bir çok kullanıcı istediği zaman rakip bulamaya bilir. Zorlu bir rakip ile oynamayı seven kullanıcının rakip bulması daha zordur. İşte sezgisel sürü zekası optimizasyon algoritmaları ile yazılmış bu hapis tavlası oyununu oyun firmaları sitelerine ekleyip güçlü bir yapay zekaya sahip hapis tavlası ile hırslı kullanıcılarına zevkli dakikalar yaşatabilir.

KAYNAKÇA

Abe, G., 1994, ”Unsolved Problems on Magic Squares.” Disc. Math. 127, 3-13

Abiyev, A., 1996, “Sayılı Sihirli Karelerin Doğal Şifresi”, Muradiye KV, Ankara

Akyol vd, 2012, Nevşehir Üniversitesi Fen Bilimleri Enstitü Dergisi 1

Alataş, B., 2007, Kaotik Haritalı Parçacık Sürü Optimizasyon Algoritmaları Geliştirme, Doktora Tezi, Fırat Üniversitesi, Fen Bilimleri Enstitüsü,

Apostolopoulos, T., Vlachos, A., 2011, Application of the Firefly Algorithm for Solving the Economic Emissions Load Dispatch Problem, Hindawi Publishing Corporation International conference Journal of Combinatorics, vol. 2011, doi:10.1155/2011/523806

Başbuğ, S., 2008, Bakteriyel Besin Arama Algoritması ile Lineer Anten Dizilerinin Diyagram Sıfırlaması, Yüksek Lisans Tezi, Erciyes Üniversitesi, Fen Bilimleri Enstitüsü

Bierman, H.S., Fernandez, L., 1993, “Game Theory with Econpmic Applications”, Addison-Wesley

Christopher, J.H., 1991, “Magic Squares and Linear Algebra”, American Mathematical Monthly, 98, No 6

Dorigo M., Maniezzo, V., Colorni, A., 1991, The Ant System: An Autocatalytic Optimizing Process. Tech. Rep. No. 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy

Erkalkan, E., 2010, Esnek Programlama Yaklaşımları ile Oyun Geliştirme, Marmara Üniversitesi Fen Bilimleri Enstitüsü Yüksek Lisans Tezi

Herik, H., Uiterwijk, J., Rijswijck, J., 2002, "Games solved: Now and in the future", Artificial Intelligence, 134, p. 277-311, Elsevier

Jiang, M., Yuan, D., Cheng, Y., 2009, Improved Artificial Fish Swarm Algorithm, Fifth International Conference on Natural Computation, 281-285

Karaboğa, D., 2011, Yapay Zekâ Optimizasyon Algoritmaları, Nobel Yayın Dağıtım

Kennedy, J., Eberhart, R. C, 1995, Particle Swarm Optimization. IEEE International Conference on Neural Networks, vol.IV, 1942-1948, Piscataway, NJ

Krishnanand, K.N., 2005, Ghose, D. Detection of Multiple Source Locations Using a Glowworm Metaphor with Applications to Collective Robotics, IEEE Swarm Intelligence Symposium, 84-91

Krishnanand, K.N., Ghose, D., 2009, Glowworm Swarm Optimisation: a New Method for Optimisin Multi-Modal Functions, International Journal of Computational Intelligence Studies, vol. 1, no. 1

Liu, C., Yan, X., Liu, C., Wu, H., 2011, The Wolf Colony Algorithm and Its Application, Chinese Journal of Electronics, vol. 20, no. 2

Lotfi, V, 1989, A Labeling Algorithm to Solve the Assignment Problem, Computers and Operations Research. (16), 397-408

Luger, G. F., Stubblefield, W.A., 1993, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 2nd ed. Benjamin/Cummings Redwood City CA

McMillan J., 1992, "Games, Strategies, and Managers", Oxford, OUP

Murty, K. G., 2003, Optimization Models For Decision Making, vol. 1, Internet Edition, Chapter 1: Models for Decision Making, 1-18

Nabiyev V.V., Pehlivan H., 2008, Towards Reverse Scheduling with Final States of Sport Disciplines, Applied and Computational Mathematics, ACM, Computational Mathematics, Vol 7, N 1, pp. 89-107 (SCI Exp)

Nabiyev, V.V., 2003, "Yapay Zeka", Seçkin Yayıncılık, Ankara, 97-98,105 106,113,127-140

Norvig, P., 1992, Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp, Morgan Kaufmann San Francisco CA

Pamukkale Üniversitesi, 2012, Mühendislik Bilimleri Dergisi, Cilt 18, Sayı 2

Pohl, I., 1997, "Practical and Teoretical considerations in heuristic search algorithms", In Elcock, E.W., and Michie, D., editors, Machine Intelligence 8, Ellis Horwood, England, pp. 55-72

Russell, S. & Norvig, P., 2003, Artificial Intelligence: A Modern Approach. New Jersey: Prentice Hall

Vagifoğlu, Prof. Dr. V.S., 2010, "Yapay Zeka: İnsan –Bilgisayar Etkileşimi", 1. Baskı

Yang X. S., 2009 Firefly Algorithms Formultimodal Optimization, Proceedings of the Stochastic Algorithms: Foundations and Applications, Lecture Notes in Computing Sciences, vol. 5792, 178-178, Springer, Sapporo, Japan

E-KAYNAKÇA

[1]http://www.elektrik.gen.tr/icerik/oyun-programlama-ve-yapay-zeka (02.03.2012)

[2]http://www.turkzeka.com/zeka/yapayzekaw.asp (02.03.2012)

[3]www.sakirkocabas.com/files/yzgir_1n.rtf (17.03.2012)

[4]http://aiguy.files.wordpress.com/2009/05/knights_tour-final.pdf (13.05.2012)

[5]http://www.hasanbalik.com/doc/yapayzeka.pdf (15.05.2012)

[6]http://www.cur-cuna.com/tr/eglence/oyun/oyun0005.html (24.05.2012)

[7] http://www.tavladayiz.com/tavla-nedir.html (13.07.2012)

EK-1: Oynanacak hamlelerin hesaplanıp yeni uzay durumunun ara yüzde gösterilmesini sağlayan metot

```
privatevoid timerBilgisayar_Tick(object sender, EventArgs e)
{
if (oyunBittiMi())
  {
    buttonYeniOyun.Enabled = true;
    buttonOyna.Enabled = false;
    buttonZarAt.Enabled = false;

    timerBilgisayar.Enabled = false;
MessageBox.Show("Oyunu kaybettiniz.");
return;
  }

if (bilgisayarTasToplamaKontrol())
    bilgisayarTaslariNormalTopla();

if (oyunBittiMi())
  {
    buttonYeniOyun.Enabled = true;
    buttonOyna.Enabled = false;
    buttonZarAt.Enabled = false;

    timerBilgisayar.Enabled = false;
MessageBox.Show("Oyunu kaybettiniz.");
return;
  }

if (kalanHamleSayisi == 0)
  {
    buttonZarAt.Enabled = true;
    timerBilgisayar.Enabled = false;
return;
  }

  oynanabilirSifirla();

  taslarKopyala();

  oynanabilirHamleler();

  oynabilirIlkDortHamle();

if (oyunBittiMi())
  {
    buttonYeniOyun.Enabled = true;
    buttonOyna.Enabled = false;
```

```
    buttonZarAt.Enabled = false;

    timerBilgisayar.Enabled = false;
MessageBox.Show("Oyunu kaybettiniz.");
return;
  }



if (HamleZar[0, 0] == 100.0)
  {
    buttonZarAt.Enabled = true;
    timerBilgisayar.Enabled = false;
    textBoxHamleler.Text += "Oynayacak hamle yok...[" + zar[0].ToString() +
"][" + zar[1].ToString() +
"]  \r\n";
return;
  }

  oyna();

goster();// elde edilen yeni uzay durumu arayüze yansıtılır
}
```

EK-2: Oynanabilecek hamleler hesaplandıktan yeni durum uzayının elde edildiği metot

```
void oyna()
{
  if (zar[0] == zar[1])
  {
    bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 0] -= 1;

    if (bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 0] == 0)
      bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 1] = 0;

    oynanacakYer = Convert.ToInt32(HamleZar[0, 1]) +
((Convert.ToInt32(HamleZar[0, 2])+1) * zar[0]);

    bilgisayarTaslari[oynanacakYer, 0] += 1;
    bilgisayarTaslari[oynanacakYer, 1] = 1;

    if (kullaniciTaslari[23 - Convert.ToInt32(oynanacakYer), 0] == 1)
      kullaniciTaslari[23 - Convert.ToInt32(oynanacakYer), 1] = 0;

    kalanHamleSayisi -= Convert.ToInt32(HamleZar[0, 2]) +1;

    if (kalanHamleSayisi == 3)
    {
      textBoxZar1.Visible = false;
      labelBir.Visible = false;
    }
    else
    if (kalanHamleSayisi == 2)
    {
      textBoxZar1.Visible = false;
      textBoxZar2.Visible = false;
      labelBir.Visible = false;
      labelIki.Visible = false;
    }
    else
    if (kalanHamleSayisi == 1)
    {
      textBoxZar1.Visible = false;
      textBoxZar2.Visible = false;
      textBoxZar3.Visible = false;
      labelBir.Visible = false;
      labelIki.Visible = false;
      labelUc.Visible = false;
    }
    else
    if (kalanHamleSayisi == 0)
    {
```

```
      textBoxZar1.Visible = false;
      textBoxZar2.Visible = false;

      textBoxZar3.Visible = false;
      textBoxZar4.Visible = false;
      labelBir.Visible = false;
      labelIki.Visible = false;
      labelUc.Visible = false;
      labelDort.Visible = false;
    }
  }
  else
  if (Convert.ToInt32(HamleZar[0, 3]) == 1)
  {
    bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 0] -= 1;

    if (bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 0] == 0)
      bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 1] = 0;

    oynanacakYer = Convert.ToInt32(HamleZar[0, 1]) + zar[0] + zar[1];

    bilgisayarTaslari[oynanacakYer, 0] += 1;
    bilgisayarTaslari[oynanacakYer, 1] = 1;

    if (kullaniciTaslari[23 - Convert.ToInt32(oynanacakYer), 0] == 1)
       kullaniciTaslari[23 - Convert.ToInt32(oynanacakYer), 1] = 0;

    kalanHamleSayisi -= 2;

    textBoxZar1.Visible = false;
    textBoxZar2.Visible = false;
    labelBir.Visible = false;
    labelIki.Visible = false;
  }
  else
  if (Convert.ToInt32(HamleZar[0, 3]) == 0)
  {
    bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 0] -= 1;

    if (bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 0] == 0)
      bilgisayarTaslari[Convert.ToInt32(HamleZar[0, 1]), 1] = 0;

    oynanacakYer = Convert.ToInt32(HamleZar[0, 1]) +
zar[Convert.ToInt32(HamleZar[0, 2])];

    bilgisayarTaslari[oynanacakYer, 0] += 1;
    bilgisayarTaslari[oynanacakYer, 1] = 1;

    if (kullaniciTaslari[23 - Convert.ToInt32(oynanacakYer), 0] == 1)
      kullaniciTaslari[23 - Convert.ToInt32(oynanacakYer), 1] = 0;
```

```
   kalanHamleSayisi -= 1;

   zarSecme = Convert.ToInt32( HamleZar[0, 2] );

   if (zarSecme == 0)
   {
     textBoxZar1.Visible = false;
     labelBir.Visible = false;
   }
   else
   if (zarSecme == 1)
   {
     textBoxZar2.Visible = false;
     labelIki.Visible = false;
   }
  }

  textBoxHamleler.Text += "Bilgisayar : " + (Convert.ToInt32(HamleZar[0, 1])
+ 1).ToString() +
                          " --> " + (oynanacakYer + 1).ToString() + "\r\n";}
```

EK-3: Bilgisayar taşlarının oynayabileceği tüm hamleleri hesaplanyan metot

```
private void oynanabilirHamleler()
{
  for(int i=0; i<25 ; i++)
  {
    if ( bilgisayarTaslari[i,0] > 0 && bilgisayarTaslari[i,1]==1)
// bilgisayar taşının orada olup olmadığına bakılıyor
    {
      if (zar[0] == zar[1])
     // 2 zarda aynıysa 4 ile çarpılmış haline bak yani  1x1  2x2  3x3  4x4  5x5
6x6
      {
        for (int l = 0; l < kalanHamleSayisi; l++)
 // seviye için kaç tanesini oynabilecek 1 2 3 4 tane mi?
        {
          if (i + zar[0] * (l + 1) < 24)
   // taşın tahta dışına çıkıp çıkmadığı kontrol ediliyor
          {
            if (kullaniciTaslari[23 - i - zar[0] * (l + 1), 0] == 0  ||
               (kullaniciTaslari[23 - i - zar[0] * (l + 1), 0] == 1 &&
                kullaniciTaslari[23 - i - zar[0] * (l + 1), 1] == 0) ||
               (kullaniciTaslari[23 - i - zar[0] * (l + 1), 0] == 1 &&
                bilgisayarTaslari[i + zar[0] * (l + 1), 0] == 0))
// rakip taş sayısı 2 den az var ise
            {
              taslarKopyala();
              bilgisayarTaslariYedek[i, 0] -=  1;


              if (bilgisayarTaslariYedek[i, 0] == 0)
                bilgisayarTaslariYedek[i, 1] = 0;


              bilgisayarTaslariYedek[i + zar[0] * (l + 1), 0] +=  1;
              bilgisayarTaslariYedek[i + zar[0] * (l + 1), 1] = 1;

              enIyiHamleyiBul();

              bilgisayarOynayabilirCiftZar[i, l, 0] = 1;

              bilgisayarOynayabilirCiftZar[i, l, 1] = kiranTekZarSayisi;
              bilgisayarOynayabilirCiftZar[i, l, 2] = kiranIkiliZarSayisi;
              bilgisayarOynayabilirCiftZar[i, l, 3] = kiranCiftiZarSayisi;
              bilgisayarOynayabilirCiftZar[i, l, 4] = (kiranTekZarSayisi *
              (Convert.ToDouble(1) / Convert.ToDouble(6))) +
              (kiranIkiliZarSayisi * (Convert.ToDouble(1) /
Convert.ToDouble(18))) +
```

```
            (kiranCiftiZarSayisi * (Convert.ToDouble(1) /
Convert.ToDouble(36)));

          }
          else
            break;

        }
       }

       continue;

     }

     for (int j=0 ; j<2 ; j++)  // zar için
     {
       if (i + zar[j] < 24) // bilgisayar taşının tahta dışına çıkıp çıkmadığı kontrol
ediliyor
   {
   if (kullaniciTaslari[23 - i - zar[j], 0] == 0  ||
           (kullaniciTaslari[23 - i - zar[j], 0] == 1 &&
            kullaniciTaslari[23 - i - zar[j], 1] == 0) ||
           (kullaniciTaslari[23 - i - zar[j], 0] == 1 &&
            bilgisayarTaslari[i + zar[j], 0] == 0)) //kullanıcı taş sayısı 2 den az var
ise
        {
    if (zarSecme == j ) // 1. hamle yapıldıktan sonra aynı zarın tekrar
oynatılmaması kontrol ediliyor
            continue;

          taslarKopyala();


          bilgisayarTaslariYedek[i, 0] -= 1;

          if (bilgisayarTaslariYedek[i, 0] == 0)
            bilgisayarTaslariYedek[i, 1] = 0;

          bilgisayarTaslariYedek[i + zar[j], 0] +=  1;
          bilgisayarTaslariYedek[i + zar[j], 1] = 1;

          enIyiHamleyiBul();

          bilgisayarOynayabilir[i, j, 0] = 1;

          bilgisayarOynayabilir[i, j, 1] = kiranTekZarSayisi;
          bilgisayarOynayabilir[i, j, 2] = kiranIkiliZarSayisi;
          bilgisayarOynayabilir[i, j, 3] = kiranCiftiZarSayisi;
```

```
            bilgisayarOynayabilir[i, j, 4] = (kiranTekZarSayisi *
            (Convert.ToDouble(1) / Convert.ToDouble(6))) +
            (kiranIkiliZarSayisi * (Convert.ToDouble(1) / Convert.ToDouble(18)))
+
            (kiranCiftiZarSayisi * (Convert.ToDouble(1) /
Convert.ToDouble(36)));

          }
        }
      }

     if (i + zar[0] + zar[1] < 24 && kalanHamleSayisi == 2) // bilgisayar taşının
iki zarın toplamı için tahta dışına çıkıp çıkmadığı kontrol ediliyor.
kalanHamleSayisi == 2 kontrolü ile hamle yaptıktan sonra tek zar kaldıysa
hamle sayısı olarak çift zar ihtimaline girmesin
     {
       if ((kullaniciTaslari[23 - i - zar[0], 0] == 0 ||
           (kullaniciTaslari[23 - i - zar[0], 0] == 1 &&
            kullaniciTaslari[23 - i - zar[0], 1] == 0) ||
           (kullaniciTaslari[23 - i - zar[0], 0] == 1 &&
            bilgisayarTaslari[i + zar[0], 0] == 0)) ||
           (kullaniciTaslari[23 - i - zar[1], 0] == 0 ||
           (kullaniciTaslari[23 - i - zar[1], 0] == 1 &&
            kullaniciTaslari[23 - i - zar[1], 1] == 0) ||
           (kullaniciTaslari[23 - i - zar[1], 0] == 1 &&
            bilgisayarTaslari[i + zar[1], 0] == 0)))
//bilgisayar taşı hareket edebiliyor mu

           if (kullaniciTaslari[23 - i - zar[0] - zar[1], 0] < 2)
// kullanıcı taş sayısı 2 den az var ise
           {
             if (bilgisayarTaslari[i + zar[0] + zar[1], 0] == 1 &&
                 bilgisayarTaslari[i + zar[0] + zar[1], 1] == 0)
               continue;

             taslarKopyala();

             bilgisayarTaslariYedek[i, 0] -=  1;

             if (bilgisayarTaslariYedek[i, 0] == 0)
               bilgisayarTaslariYedek[i, 1] = 0;

             bilgisayarTaslariYedek[i + zar[0] + zar[1], 0] +=  1;
             bilgisayarTaslariYedek[i  + zar[0] + zar[1], 1] = 1;

             enIyiHamleyiBul();

             bilgisayarOynayabilirIkiZar[i, 0] = 1;

             bilgisayarOynayabilirIkiZar[i, 1] = kiranTekZarSayisi;
```

```
            bilgisayarOynayabilirIkiZar[i, 2] = kiranIkiliZarSayisi;
            bilgisayarOynayabilirIkiZar[i, 3] = kiranCiftiZarSayisi;
            bilgisayarOynayabilirIkiZar[i, 4] = (kiranTekZarSayisi *
            (Convert.ToDouble(1) / Convert.ToDouble(6))) +
            (kiranIkiliZarSayisi * (Convert.ToDouble(1) /
Convert.ToDouble(18))) +
            (kiranCiftiZarSayisi * (Convert.ToDouble(1) /
Convert.ToDouble(36)));

        }
    }
   }
  }
}
```

EK-4: Bilgisayar taşının bulunduğu hane zar oynama durumuna göre derecelendirilip kullanıcı tarafından hapis edilebilme ihtimalini hesaplayan metot

```
private void hamleSayisi(int i)
{
 for(int j=0; j<24; j++)  // kullanıcının taşları dolaşılıyor
 {
  if (kullaniciTaslari [j,0] > 0 && kullaniciTaslari [j,1]==1) // kullanıcının
  taşının orada olup olmadığına bakılıyor
  {
   for (int k=1 ; k<7; k++) // toplam tek zarın sayısı
   {
    if (23 - j - k < 0) //bilgisayar taşının tahtanın dışına çıkıp çıkmadığı
kontrol ediliyor
      continue;

    if (bilgisayarTaslari[23 - j - k, 0] > 1) //ilk zarın hareket edip
edemeyeceğine bakılıyor
      continue;

    if (bilgisayarTaslari[23 - j - k, 0] == 1 &&
       bilgisayarTaslari[23 - j - k, 1] == 1 &&
       kullaniciTaslari[j + k , 0] == 1) // ilk zarın hareket ettiği yerde hem
bilgisayar hem kullanıcının taşından 1 tane olup bilgisayar üstteyse oraya taş
koyamayız
        continue;

    if ( i == 23 -  j - k  ) // hapis edilebileceğk zar hamleleri kontrol ediliyor
   kiranTekZarSayisi ++;

    for (int l=1 ; l<7; l++) // toplam ikinci zarın sayısı
    {
     if (k != l)
       if ( i  ==  23 -  j – k - l )// hapis edilebilecek zar hamleleri kontrol
ediliyor
      kiranIkiliZarSayisi++;
    }

    for  (int l=1;l<5;l++) // 1x1 2x2 3x3 4x4 5x5 6x6 gibi olanların 1 2 3 4
tane
    aynı taşı hareket ettirince hapis edebiliyor mu kontrol ediliyor
    {
     if (23 - j - (k * l) < 0) //bilgisayar taşının tahtanın dışına çıkıp çıkmadığı
kontrol ediliyor
       continue;

     if (bilgisayarTaslari[23 - j - (k * l), 0] < 2) // zarın hareket edip
```

```
        edemeyeceğine bakılıyor
        {
          if (i == 23 - j - (k * l)) // hapis edilebilecek çift zar hamleleri kontrol
          ediliyor
            kiranCiftiZarSayisi++;
        }
        else //  eğer serideki taş hareket edemiyorsa bir sonraki de
oynayamayacağı
        için iptal ediliyor
          break;

       }
     }
   }
  }
}
```

EK-5: Bilgisayarın hamle olasılıklarından en iyi dört hamlesini hesaplayan metot

```
void oynabilirIlkDortHamle()
{
  for (int i = 0; i < 24; i++)
  {
    for (int j = 0; j < 2; j++)
    {
      hamleSirala(i, j, 0);
    }
    for (int j = 0; j < 4; j++)
    {
      hamleSirala(i, j, -1);
    }
    hamleSirala(i,-1, 0);
  }
}
```

EK-6: Bilgisayar taşlarının hapis edilebileceği olasılığı sıralayan metot

```
private void hamleSirala(int i, int j,int k) // tek mi, cift mi, ikili mi zar
oynanacak
{
 if (j == -1)
 {
  if (HamleZar[0, 0] > bilgisayarOynayabilirIkiZar[i, 4] &&
     bilgisayarOynayabilirIkiZar[i, 0] == 1)
 hamleDoldur(i, j, 0, 1, bilgisayarOynayabilirIkiZar[i, 4]);
  else
  if (HamleZar[1, 0] > bilgisayarOynayabilirIkiZar[i, 4] &&
     bilgisayarOynayabilirIkiZar[i, 0] == 1)
       hamleDoldur(i, j, 1, 1, bilgisayarOynayabilirIkiZar[i, 4]);
  else
  if (HamleZar[2, 0] > bilgisayarOynayabilirIkiZar[i, 4] &&
     bilgisayarOynayabilirIkiZar[i, 0] == 1)
       hamleDoldur(i, j, 2, 1, bilgisayarOynayabilirIkiZar[i, 4]);
  else
  if (HamleZar[3, 0] > bilgisayarOynayabilirIkiZar[i, 4] &&
     bilgisayarOynayabilirIkiZar[i, 0] == 1)
       hamleDoldur(i, j, 3, 1, bilgisayarOynayabilirIkiZar[i, 4]);
 }
 else
 if (k == -1)
 {
  if (HamleZar[0, 0] > bilgisayarOynayabilirCiftZar[i, j, 4] &&
     bilgisayarOynayabilirCiftZar[i, j, 0] == 1)
       hamleDoldur(i, j, 0, 2, bilgisayarOynayabilirCiftZar[i, j, 4]);
  else
  if (HamleZar[1, 0] > bilgisayarOynayabilirCiftZar[i, j, 4] &&
     bilgisayarOynayabilirCiftZar[i, j, 0] == 1)
     hamleDoldur(i, j, 1, 2, bilgisayarOynayabilirCiftZar[i, j, 4])
  else
  if (HamleZar[2, 0] > bilgisayarOynayabilirCiftZar[i, j, 4] &&
     bilgisayarOynayabilirCiftZar[i, j, 0] == 1)
       hamleDoldur(i, j, 2, 2, bilgisayarOynayabilirCiftZar[i, j, 4]);
  else
  if (HamleZar[3, 0] > bilgisayarOynayabilirCiftZar[i, j, 4] &&
     bilgisayarOynayabilirCiftZar[i, j, 0] == 1)
       hamleDoldur(i, j, 3, 2,bilgisayarOynayabilirCiftZar[i, j, 4]);
 }
 else
 {
  if (HamleZar[0, 0] > bilgisayarOynayabilir[i, j, 4] &&
     bilgisayarOynayabilir[i, j, 0] == 1)
       hamleDoldur(i, j, 0, 0, bilgisayarOynayabilir[i, j, 4]);
  else
  if (HamleZar[1, 0] > bilgisayarOynayabilir[i, j, 4] &&
```

```
        bilgisayarOynayabilir[i, j, 0] == 1)
          hamleDoldur(i, j, 1, 0, bilgisayarOynayabilir[i, j, 4]);
      else
      if (HamleZar[2, 0] > bilgisayarOynayabilir[i, j, 4] &&
        bilgisayarOynayabilir[i, j, 0] == 1)
          hamleDoldur(i, j, 2, 0, bilgisayarOynayabilir[i, j, 4]);
      else
      if (HamleZar[3, 0] > bilgisayarOynayabilir[i, j, 4] &&
        bilgisayarOynayabilir[i, j, 0] == 1)
          hamleDoldur(i, j, 3, 0, bilgisayarOynayabilir[i, j,  4]);
    }
  }
```

EK-7: Zar ve hane değerine göre olasılıkları diziye atan metot

```
private void  hamleDoldur(int i,int j, int l,int tip,double deger)
{
  for (int m = 3; m > l; m--)
  {
    HamleZar[m, 0] = HamleZar[m - 1, 0];
    HamleZar[m, 1] = HamleZar[m - 1, 1];
    HamleZar[m, 2] = HamleZar[m - 1, 2];
    HamleZar[m, 3] = HamleZar[m - 1, 3];
  }

  HamleZar[l, 0] = deger;
  HamleZar[l, 1] = i;
  HamleZar[l, 2] = j;
  HamleZar[l, 3] = tip;
}
```

EK-8: Hamlenin hapis edilebilebilme ihtimalini hesaplayan metot

```
void enIyiHamleyiBul()
{
  kiranTekZarSayisi =0;
  kiranIkiliZarSayisi=0;
  kiranCiftiZarSayisi=0;
  for (int i = 0; i < 24; i++)
  {
    if (bilgisayarTaslariYedek[i, 0] == 1 &&
        bilgisayarTaslariYedek[i, 1] == 1 &&
        kullaniciTaslari[23-i,0] != 1) //orada tek taş var fakat kullanıcın taşını
hapis edildiyse bunu kullanıcı hapis edilemez
    {
      hamleSayisi(i);
    }
  }
}
```

EK-9: Toplanacak hamlenin kotrolünü yapan metot

```
private bool bilgisayarTasToplamaKontrol()
{
  for (int i = 0; i < 18; i++)
  {
    if (bilgisayarTaslari[i, 0] > 0)
      return false;
  }

  for (int i = 18; i <24 ; i++)
  {
    if (bilgisayarTaslari[i, 0] == 1 && bilgisayarTaslari[i, 1] == 0)
      return false;
  }

  return true;
}
```

EK-10: Toplanması gereken taşların belirlendiği metot

```
private void bilgisayarTaslariNormalTopla()
{
  //toplanacak zar yerinde var ise toplamaya başla
  if (textBoxZar1.Visible == true)
  {
    nereden = Convert.ToInt32(textBoxZar1.Text);

    if (bilgisayarTaslari[24 - nereden, 0] > 0)
    {
      kalanHamleSayisi--;
      bilgisayarTaslari[24 - nereden, 0]--;
      textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
      textBoxZar1.Visible = false;
      labelBir.Visible = false;
      zarSecme = 0;
      textBoxHamleler.Text += "Bilgisayar : " + nereden.ToString() + " toplandı
\r\n";

      if (bilgisayarTaslari[24 - nereden, 0] == 0)
      {
        bilgisayarTaslari[24 - nereden, 1] = 0;
        if (kullaniciTaslari[nereden - 1, 0] == 1)
          kullaniciTaslari[nereden - 1, 1] = 1;
      }
    }
  }

  if (textBoxZar2.Visible == true)
  {
    nereden = Convert.ToInt32(textBoxZar2.Text);

    if (bilgisayarTaslari[24 - nereden, 0] > 0)
    {
      kalanHamleSayisi--;
      bilgisayarTaslari[24 - nereden, 0]--;
      textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
      textBoxZar2.Visible = false;
      labelIki.Visible = false;
      zarSecme = 1;
      textBoxHamleler.Text += "Bilgisayar : " + nereden.ToString() + " toplandı
\r\n";

      if (bilgisayarTaslari[24 - nereden, 0] == 0)
      {
        bilgisayarTaslari[24 - nereden, 1] = 0;
```

```
        if (kullaniciTaslari[nereden - 1, 0] == 1)
          kullaniciTaslari[nereden - 1, 1] = 1;
      }
    }
  }

  if (textBoxZar3.Visible == true)
  {
    nereden = Convert.ToInt32(textBoxZar3.Text);
    if (bilgisayarTaslari[24 - nereden, 0] > 0)
    {
      kalanHamleSayisi--;
      bilgisayarTaslari[24 - nereden, 0]--;
      textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
      textBoxZar3.Visible = false;
      labelUc.Visible = false;
      textBoxHamleler.Text += "Bilgisayar : " + nereden.ToString() + " toplandı
\r\n";

      if (bilgisayarTaslari[24 - nereden, 0] == 0)
      {
        bilgisayarTaslari[24 - nereden, 1] = 0;
        if (kullaniciTaslari[nereden - 1, 0] == 1)
          kullaniciTaslari[nereden - 1, 1] = 1;
      }
    }
  }

  if (textBoxZar4.Visible == true)
  {
    nereden = Convert.ToInt32(textBoxZar4.Text);

    if (bilgisayarTaslari[24 - nereden, 0] > 0)
    {
      kalanHamleSayisi--;
      bilgisayarTaslari[24 - nereden, 0]--;
      textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
      textBoxZar4.Visible = false;
      labelDort.Visible = false;
      textBoxHamleler.Text += "Bilgisayar : " + nereden.ToString() + " toplandı
\r\n";

      if (bilgisayarTaslari[24 - nereden, 0] == 0)
      {
        bilgisayarTaslari[24 - nereden, 1] = 1;
        if (kullaniciTaslari[nereden - 1, 0] == 1)
          kullaniciTaslari[nereden - 1, 1] = 1;
      }
```

```
    }
  }

  // arkasında taş yok önünde taş var ise topla
  toplaKontrol = true;
  if (textBoxZar1.Visible == true)
  {
    nereden = Convert.ToInt32(textBoxZar1.Text);

    for (int i = 18; i < 24 - nereden; i++)
    {
      if (bilgisayarTaslari[i, 0] > 0)
      {
        toplaKontrol = false;
        break;
      }
    }

    if (toplaKontrol)
    {
      for (int i = 24- nereden; i < 24; i++)
      {
        if (bilgisayarTaslari[i, 0] > 0)
        {
          kalanHamleSayisi--;
          bilgisayarTaslari[i, 0]--;
          textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
          textBoxZar1.Visible = false;
          labelBir.Visible = false;
          zarSecme = 0;
          textBoxHamleler.Text += "Bilgisayar : " + i.ToString() + " toplandı
\r\n";

          if (bilgisayarTaslari[i, 0] == 0)
          {
            bilgisayarTaslari[i, 1] = 0;
            if (kullaniciTaslari[23 - i, 0] == 1)
              kullaniciTaslari[23 - i, 1] = 1;
          }
          break;
        }
      }
    }
  }


  toplaKontrol = true;
  if (textBoxZar2.Visible == true)
  {
```

```
    nereden = Convert.ToInt32(textBoxZar2.Text);
    for (int i = 18; i < 24 - nereden; i++)
    {
     if (bilgisayarTaslari[i, 0] > 0)
     {
      toplaKontrol = false;
      break;
     }
    }

    if (toplaKontrol)
    {
     for (int i = 24 - nereden; i < 24; i++)
     {
      if (bilgisayarTaslari[i, 0] > 0)
      {
       kalanHamleSayisi--;
       bilgisayarTaslari[i, 0]--;
       textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
       textBoxZar2.Visible = false;
       labelIki.Visible = false;
       zarSecme = 1;
       textBoxHamleler.Text += "Bilgisayar : " + i.ToString() + " toplandı
\r\n";
       if (bilgisayarTaslari[i, 0] == 0)
       {
        bilgisayarTaslari[i, 1] = 0;
        if (kullaniciTaslari[23 - i, 0] == 1)
         kullaniciTaslari[23 - i, 1] = 1;
       }

       break;
      }
     }
    }
   }
   toplaKontrol = true;
   if (textBoxZar3.Visible == true)
   {
    nereden = Convert.ToInt32(textBoxZar3.Text);

    for (int i = 18; i < 24 - nereden; i++)
    {
     if (bilgisayarTaslari[i, 0] > 0)
     {
      toplaKontrol = false;
      break;
     }
    }
```

```
    if (toplaKontrol)
    {
      for (int i = 24 - nereden; i < 24; i++)
      {
        if (bilgisayarTaslari[i, 0] > 0)
        {
          kalanHamleSayisi--;
          bilgisayarTaslari[i, 0]--;
          textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
          textBoxZar3.Visible = false;
          labelUc.Visible = false;
          textBoxHamleler.Text += "Bilgisayar : " + i.ToString() + " toplandı
\r\n";

          if (bilgisayarTaslari[i, 0] == 0)
          {
            bilgisayarTaslari[i, 1] = 0;
            if (kullaniciTaslari[23 - i, 0] == 1)
              kullaniciTaslari[23 - i, 1] = 1;
          }

          break;
        }
      }
    }
  }

  toplaKontrol = true;
  if (textBoxZar4.Visible == true)
  {
    nereden = Convert.ToInt32(textBoxZar4.Text);
    for (int i = 18; i < 24 - nereden; i++)
    {
      if (bilgisayarTaslari[i, 0] > 0)
      {
        toplaKontrol = false;
        break;
      }
    }

    if (toplaKontrol)
    {
      for (int i = 24 - nereden; i < 24; i++)
      {
        if (bilgisayarTaslari[i, 0] > 0)
        {
          kalanHamleSayisi--;
          bilgisayarTaslari[i, 0]--;
```

```
        textBoxToplaBilgisayar.Text =
(Convert.ToInt32(textBoxToplaBilgisayar.Text) + 1).ToString();
        textBoxZar4.Visible = false;
        labelDort.Visible = false;
        textBoxHamleler.Text += "Bilgisayar : " + i.ToString() + " toplandı
\r\n";

        if (bilgisayarTaslari[i, 0] == 0)
        {
          bilgisayarTaslari[i, 1] = 0;
          if (kullaniciTaslari[23 - i, 0] == 1)
            kullaniciTaslari[23 - i, 1] = 1;
        }

        break;
       }
      }
    }
  }
  goster();
}
```

EK-11: Oyunun kim tarafından kazanıldığını ve kaç puanla kazanıldığını hesaplayan metot

```
private bool oyunBittiMi()
{
  kalanTasSayisiBilgisayar = Convert.ToInt32( textBoxToplaBilgisayar.Text );
  kalanTasSayisiKullanici = Convert.ToInt32(textBoxToplaKullanici.Text);
  if (kalanTasSayisiKullanici == 15)
  {
    if (kalanTasSayisiBilgisayar == 0)
    {
      labelSkorKullanici.Text = ( Convert.ToInt32(labelSkorKullanici.Text) + 2
).ToString();
      textBoxHamleler.Text += "Tebrikler oyunu 2 puan ile kazandınız.\r\n";
      return true;
    }
    textBoxHamleler.Text += "Tebrikler oyunu 1 puan ile kazandınız.\r\n";
    labelSkorKullanici.Text = (Convert.ToInt32(labelSkorKullanici.Text) +
1).ToString();
    return true;
  }

  if (kalanTasSayisiBilgisayar == 15)
  {
    if (kalanTasSayisiKullanici == 0)
    {
      labelSkorBilgisayar.Text = (Convert.ToInt32(labelSkorBilgisayar.Text) +
2).ToString();
      textBoxHamleler.Text += "Oyunu 2 puan ile kaybettiniz.\r\n";
      return true;
    }

    textBoxHamleler.Text += "Oyunu 1 puan ile kaybettiniz.\r\n";
    labelSkorBilgisayar.Text = (Convert.ToInt32(labelSkorBilgisayar.Text) +
1).ToString();
    return true;
  }
  return false;
}
```

EK-12: Dizilerin sıfırlandğı metot

```
private void oynanabilirSifirla()
{
  for(int i=0; i<25 ; i++)
  {
    bilgisayarOynayabilirIkiZar[i, 0] = 0;
    for (int l = 0; l < 2; l++)
    {
      bilgisayarOynayabilir[i, l,0] = 0;
    }
    for (int k = 0; k < 4; k++)
      bilgisayarOynayabilirCiftZar[i, k,0] = 0;
  }
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 4; j++)
    {
      HamleZar[i, j] = 100;
    }
  }
}
```

EK-13: Mevcut durum uzayını ara tabloya kopyalayan metot

```
private void taslarKopyala()
{
  for (int i = 0; i < 24; i++)
  {
    for (int j  = 0; j  <2; j ++)
    {
      bilgisayarTaslariYedek[i, j] = bilgisayarTaslari[i,j];
    }
  }
}
```

ÖZGEÇMİŞ

Hakan ŞAHİN
Bahçelievler / İstanbul
Cep:(505) 453 94 67 – (505) 540 04 67
E-mail: hakansahin_18@hotmail.com

KİŞİSEL BİLGİLER

Uyruğu: T.C

Doğum Yeri : İstanbul

Doğum Tarihi : 18/11/1986

Medeni Durum : Bekar

Askerlik Durumu : 08/2007 – 01/2008 tarihleri arasında kısa dönem olarak yaptım

İŞ / KARİYER HEDEFİ

2000 yılında başlayan bilgisayar merakım sayesinde sürekli bilgisayarla iç içe yaşamaya ve bu alandaki bilgilerimi her zaman genişletmeye özen gösterdim. Bundan dolayı Sakarya Üniversitesi - Bilgisayar Sistemleri Öğretmenliği Bölümünden mezun oldum.

Amacım bilgisayar dünyasında söz sahibi olan bir iş yerinde çalışmakla birlikte algoritmamın güçlü olması ve içimdeki başarma azmi sayesinde bilgisayar dünyası adına insanoğlu için yararlı bir şeyler yapmak.

EĞİTİM DURUMU

2010 - : Bilgisayar Mühendisliği, İstanbul Aydın Üniversitesi (Yüksek Lisans)

2003 - 2007: Bilgisayar Sistemleri Öğretmenliği, Sakarya Üniversitesi

2000 - 2003: Zeytinburnu Teknik Lisesi (Bilgisayar Yazılım Bölümü)

BİLGİSAYAR PROGRAMLAMA DİLLERİ

- C#,
- Axapta (X++)
- Visual Basic
- Progress
- OpenEnge (4GL)
- Java
- Oracle Discovery
- Html
- XML
- Oracle ERP
- Oracle
- SSIS
- SQL (MySQL, SQL Server, SQL CE)
- Asp.NET
- Windows CE 5,6
- 8051 Mikrodenetleyici
- Asp
- Web Service
- Xcode (ObjectiveC dili ile Iphone uygulamaları geliştirmek – başlangıç seviyesi)
- Silverlight (Windows Phone 7 uygulamaları geliştirmek – başlangıç seviyesi)
- SharePoint

İŞ DENEYİMLERİM

Fiba Holding(03.2012 -)

-Kıdemli Yazılım Uzmanı

Eren Holding (04.2011-03.2012)

-Yazılım Müdür Yardımcısı

Atasay Holding (02.2009 -04.2011)

- Yazılım Uzmanı

Aras Holding (02.008 – 02.2009)

- Yazılım Yönetmeni

PROJELER

Fiba Holding(03.2012 -): Axapta ERP programının standart tüm modüllerinde kod geliştirip, standart dışı olan birçok modülü Axapta ERP ye entegre ettik. Bunlar lojistik firması aracılığı ile ürün stoklama ve bu ürünlerin yurt dışı ve mağazalar arasındaki dağıtımını sağlayan sistemin web servisleri ile Axapta'ya entegrasyonu, E-ticaret sisteminin Axapta'ya entegrasyonu,El terminallerinin yazılımının geliştirilmesi ve bu sistemin Axapta'ya entegrasyonu, bünyesindeki markaların satışlarınıdan sorumlu olduğu ülkelerde kullanılan sistemlerin Axapya'ya entegrasyonu.

Eren Holding (04.2011 – 03.2012): Oracle ERP programının birçok modülünde kod geliştirip üretim kısmındaki makinalar ile entegrasyon çalışmaları yaptım.

Atasay Holding (02.2009 – 04.2011): Progress ve OpenEdge (4GL) kullanarak RHINO adı verileren ERP programının birçok modülünde kod geliştirdim.

Aras Holding (02.008 – 02.2009): Oracle ve C# kullanılarak yazılan ESAS adı verilen ERP programının birçok modülünde kod geliştirdim.

2007:Sıcaklık ölçen sensörden alınan analog sinyal elektronik devreler ile dijital sinyale çevrildi. Bu çevrilen sinyal 8051 mikrodenetleyicisi kullanarak bilgisayarın seri portunu dinleyen C# dili ile yazılmış programa gönderilip ortamın sıcaklığı ekranda gösterildi.

2007: Bitirme tezi: Ms SQL Server 2005 ve C#'ı kullanarak ağ üzerinden çalışan, barkod okuyuculu ve stok bilgisine göre kullanıcıyı LED ler aracılığıyla uyaran kitapevi programı

2001: Zeytinburnu Teknik Lisesine kütüphane programı (Visual Basic 6.0)

SERTİFAKALAR

- **Veritabanı Yönetmeni**(Progress Software Türkiye)
- **4GL ESSENTIALS** (Progress Software Türkiye)
- **Advedced Business Language**(Progress Software Türkiye)
- **Smart Objects**(Progress Software Türkiye)

İLGİ ALANLARI

Bilgisayar, Müzik, Resim

REFERANSLAR

Doç.Dr. Ahmad BABANLI

İstanbul Aydın Üniversitesi - Doçent Doktor

Tel : 05396523934

Mert SUN

Aras Holding - Yazılım Müdürü

Tel : 05324900082

Yrd.Doç.Dr. Fahri VATANSEVER

Uludağ Üniversitesi - Doçent Doktor

Tel : 05363366080

E-Posta : fahriv@hotmail.com

ÖZET

Yüksek Lisans Tezi

HAPİS TAVLASI OYUN YAZILIMININ GELİŞTİRİMİ

Hakan ŞAHİN

Danışman: Doç. Dr. Ahmet BABANLI
Ocak, 2013

Yapay Zeka, günümüzde bilgisayar bilimlerinin en gözde dallarından biridir ve yapay zeka bilim dalı, makinelerin zeki davranmalarını sağlamaya çalışarak onların daha çok ve daha çeşitli sorunlarla tek başlarına başa çıkmalarını sağlar. Şüphesiz ki günümüzde yapay zekanın gelişmesindeki payı en yüksek olan sektörlerden biri ise bilgisayar oyunları sektörüdür. Günümüzdeki bilgisayar oyunları dünya satranç şampiyonlarını ve dama ustalarını bile yenebilmektedir. Bu motivasyon ile yola çıkılan bu çalışmada öncelikle yapay zekanın tanımı irdelenmiş, oyunlarda kullanılan yapay zekaya yardımcı arama metodları ortaya konmuş ve bu yöntemler doğrultusunda bir oyun geliştirilmiştir.

Anahtar Sözcükler: Yapay Zeka, Sezgisel Algoritma, Bilgisayar Oyunları,Hapis Tavlası Oyunu, Derinlik Öncelikli Arama

ABSTRACT

Master Thesis

BACKGAMMON PRISON GAME SOFTWARE DEVELOPMENT

Hakan ŞAHİN

Supervisor: Assoc. Prof. Dr. Ahmet BABANLI
January, 2013

Artificial Intelligence is one of the most popular branches of the computer science and it aims to make machines act intelligently, rendering them able to cope with more in number and more complex problems by themselves. Without a doubt, one of the most active sectors which aid artificial intelligence development today is the video game sector, which created programs that can be at world chess champion sand checkers masters. This work is a quest motivated by these causes, which first identifies artificial intelligence, then explain the computation methods that aid artificial intelligence science and a game has been programmed in line with these methods.

Keywords: Artificial Intelligence, Heuristic algorithm, Computer Games, Prison Backgammon Problem, Depth-First Search