KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY ZEKÂ YÖNTEMLERİYLE OYUN GELİŞTİRME

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Gökalp GÜRBÜZER

Anabilim Dalı: Bilgisayar Mühendisliği

Danışman: Doç. Dr. Adnan KAVAK

KOCAELİ, 2008

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY ZEKÂ YÖNTEMLERİYLE OYUN GELİŞTİRME

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisi Gökalp GÜRBÜZER

Tezin Enstitüye Verildiği Tarih: 26 Mayıs 2008
Tezin Savunulduğu Tarih: 08 Temmuz 2008

Tez Danışmanı	Üye	Üye
Doç. Dr. Adnan KAVAK	Yrd. Doç. Dr. H. Engin DEMİRAY	Yrd. Doç. Dr. Mustafa TÜRKBOYLARI
(.....)	(.....)	(.....)

KOCAELİ, 2008

ÖNSÖZ ve TEŞEKKÜR

Makinelerin karşılaştıkları ve daha önceden görmemiş oldukları sorunları çözmesini amaçlayan Yapay Zekâ, 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 zekâ 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 zekâya olanak sağlayacak donanımların üretilmesini sağlamış ve modern yapay zekânın ilerlemesinde büyük katkıda bulunmuştur. Günümüzde, en kolay ve en etkin olarak programlanabilen aygıtlar bilgisayarlar olduğu için yapay zekâ bir bilgisayar bilimi olmuştur ve halen pek çok uygulamada etkin olarak kullanılmaktadır.

Yapay zekâ yöntemleriyle oyun geliştirme tezimin tamamlanması sürecinde; yöntem kuramlarının çeşitlendirilmesi ve tamamlanmasında çok değerli katkısı olan ağabeyim Y. Müh. Gökhan GÜRBÜZER'e, tezin yönlendirilmesinde desteğini esirgemeyen tez danışmanım sayın Doç. Dr. Adnan KAVAK'a, tez çalışmalarım için daha fazla zaman ayırmamı sağlayan patronum sayın Dr. Rıza Can BERKAN'a ve yöneticim Altuğ Bilgin ALTINTAŞ'a teşekkürü borç bilirim. Ayrıca her zaman yanımda olan, beni bugüne yetiştirmiş bulunan annem Nilgün GÜRBÜZER ve babam Kürşat GÜRBÜZER'e sonsuz minnet duygularımı sunarım.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER

AC: Yaratığın zırh seviyesi, ([-10, 10] aralığında)
HP: Yaratığın mevcut can puanları,
LS: Least Squares, en düşük kareler,
SEU: Subjective Expected Utility, öznel beklenen yarar,
THAC0: 0 AC'li bir başka yaratığa vurmak için atılması gereken en düşük zar

YAPAY ZEKÂ YÖNTEMLERİYLE OYUN GELİŞTİRME

Gökalp GÜRBÜZER

Anahtar Kelimeler: Yapay Zekâ, Makine Öğrenmesi, Oyun Zekâsı, Yapay Zekâ Eğitimi

Özet: Yapay Zekâ, günümüzde bilgisayar bilimlerinin en gözde dallarından biridir ve yapay zekâ bilim dalı, makinelerin zeki davranmalarını sağlamaya çalışarak onların daha çok ve daha çeşitli sorunlarla tek başlarına başa çıkmalarını sağlar. Şüphesiz ki günümüzde yapay zekânın 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 zekânın tanımı irdelenmiş, yapay zekâya yardımcı hesaplama yöntemleri ortaya konmuştur. Bu hesaplama yöntemlerinden MiniMax, Tic Tac Toe adlı sıfır toplam, sıra tabanlı ve tam bilgili oyun için; eğitim yöntemlerinden Hata Güdümlü Eğitim ise sıfır olmayan toplamlı, sıra tabanlı ve eksik bilgili bir oyun olan RPG olmak üzere, iki adet farklı oyun programlanarak günümüzdeki ya da belki yarınki bir oyunda programlanan yapay zekânın ne gibi özellikler göstermesi gerektiği ortaya konmaya çalışılmıştır.

GAME PROGRAMMING USING ARTIFICIAL INTELLIGENCE METHODS

Gökalp GÜRBÜZER

Keywords: Artificial Intelligence, Machine Learning, Game Intelligence, Artificial Intelligence Training

Abstract: 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 beat world chess champions and 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. The work then states two AI programmed games; which include Tic Tac Toe, a zero-sum, turn-based, full-information game and RPG, which is a non-zero-sum, turn-based, imperfect-information game based on Tolkien's world and tries to understand what an AI should be like in a video game of today and perhaps tomorrow.

1 GİRİŞ

1.1 Yapay Zekâ'nın Tanımı

1.1.1 Zekâ'nın tanımı

Yapay Zekâ'yı tanımlamadan önce zekâ kavramının tanımına bakmak yararlı olur. Her ne kadar sık kullanılan ve kulağa basit gelen bir kavram olsa da zekâ, tanımlaması güç ve pek çok bilim dalından pek çok otoritenin birbirinden farklı tanımladığı bir kavramdır.

John McCarthy tarafından yapılan tanıma göre "Zekâ, yaşamdaki amaçlara ulaşma yetisinin hesapsal yanıdır". Yapay Zekâ bilimi bu tanımdan yola çıkarak zekânın hesapsal tabanını araştırarak çözmeye çalışır. Ancak yine John McCarthy'nin belirttiği gibi "Zekâ, kendi içinde –tamamı henüz anlaşılamamış- mekanizmalar barındırır" bu yüzden Yapay Zekâ'nın çözümünü araştırdığı her sorunun yanıtı aynı olmamaktadır ve farklı çözümler ile aşılmaya çalışılır. [1]

1.1.2 Yapay Zekâ'nın tanımı

Yapay Zekâ'nın tek bir kesin tanımı bulunmamakla birlikte ad babası olan John McCarthy 1956'da Yapay Zekâ'yı "Zeki makineler, özellikle de zeki bilgisayar programları yapma bilimi ve mühendisliğidir" diyerek tanımlamıştır. Ancak yine John McCarthy'ye göre "Benzer bir iş olan 'bilgisayarlar aracılığı ile insan zekâsını anlamaya çalışmayla' ilgili olmasına rağmen kendisini sadece biyolojik olarak gözlemlenebilen yöntemler ile sınırlandırmaz". [1]

1.1.3 Zeki Makinenin tanımı

1.1.3.1 Turing sınavı

Bir makinenin zeki olma kavramını ilk kez tanımlayan kişi Alan Turing'dir. Turing'e göre bir makinenin zeki sayılabilmesi için makine; birisi kendisi, ikisi insan olan üç oyunculu bir sınavı vermesi gerekir. İnsan oyunculardan birisi jüri olur ve ayrı ayrı her iki oyuncuyla da – hangisinin insan hangisinin bilgisayar olduğunu bilmeden - etkileşimde bulunur. Eğer jüri oyuncu, etkileşimde bulunduğu oyuncuların hangisinin insan hangisinin makine olduğunu ayırt edemezse makine sınavı vermiş olur ve zeki bir makine olarak nitelendirilebilir. [2]

Bu sınava yapay zekâ yazınında "Turing Sınavı" olarak adlandırılır. Turing sınavının tam bir zeki makine tanımı yapabildiğine ilişkin farklı görüşler vardır. Zira Turing makinesi bir zekâ tanımından yola çıkmak yerine bir makineyi bir insanla karşılaştırdığı için gerçek zekânın değil insana benzerliğin sınavı olduğu aşikârdır. Yine de ilk somut tanımı Turing vermiştir ve bütün yapay zekâ bilim çevreleri tarafından bilinen bir sınavdır.

1.2 Yapay Zekâ'nın Kullanım Alanları

Yapay zekânın kullanım alanları her geçen gün git gide artmaktadır. Bazı kullanım alanları;

- Örüntü tanıma
  - Optik karakter tanıma
  - El yazısı tanıma
  - Konuşma tanıma
  - Yüz tanıma
- Yapay yaratıcılık
- Bilgisayar görüşü, sanal gerçeklik ve görüntü işleme
- Yapay zekâ sınama
- Oyun kuramı ve stratejik planlama
- Oyun yapay zekâsı ve bilgisayar oyuncular

- Doğal dil işleme, çeviri ve sohbet botları
- Doğrusal olmayan kontrol ve robotik
- Yapay yaşam
- Otomatikleştirilmiş kavrama (Automated resoning)
- Otomasyon
- Biyolojiden ilham alınmış hesaplama
- Kavram madenciliği
- Veri madenciliği
- Bilgi temsili
- Anlam tabanlı internet
- E-posta çöplük süzmesi
- Robotik
  - Davranış tabanlı robotik
  - Kavrayıcı
  - Sibernetik
  - Evrimsel robotik
- Karma zekâ sistemi
- Zeki ajan
- Zeki kontroldür. [3]

1.3 Yapay Zekâ'da Eğitim ve Amacı

Yapay zekâda eğitimin amacı, belirli sorunlarla baş etmek üzere tasarlanmış olan yapay zekânın kendini geliştirmesi, ileride karşısına çıkabilecek daha farklı durumlarda da doğru adımlar izleyebilmesini sağlamaktır. Genel anlamda yapay zekâ eğitimi, birer minimizasyon ya da maksimizasyon sorunlarıdır ve optimizasyon kuramlarından bolca yararlanır.

Yapay zekânın eğitilmesi, sistemin kendi kendine daha gelişmiş bir yapay zekâ ortaya koymasını sağlar. Öğrenen makineler, öğrenmeyenlere göre daha fazla sayıda ve çeşitte sorunlarla baş edebilir.

1.4 Kullanılan Yapay Zekâ

Bu tezde, iki adet yapay zekâlı oyun bulunmaktadır. Birisi eski bir zekâ oyunu olan Tic Tac Toe, bir diğeri ise Tolkien dünyasından ortaya çıkmış bir oyun türünden esinlenmiş RPG'dir.

1.4.1 Tic tac toe

Tic Tac Toe oyunu, sıra tabanlı bir tahta oyunudur. İki oyuncu da kendi sırası geldiğinde kendi işaretini tahtanın istediği yerine yerleştirir. İşaretleriyle düz bir sıra yapan oyuncu oyunun galibi olur.

Tic Tac Toe yazılımında, John von Neumann'ın ortaya koyduğu MiniMax yöntemi kullanılmıştır.

1.4.2 RPG

RPG, Role Playing Game sözcüklerinin kısaltmasıdır ve aslında bir oyun değil, bir oyun türü adıdır. Oyunda karşı karşıya gelen iki düşman grup, birbirlerini ellerindeki silahlarıyla öldürmeye çalışmaktadırlar. Bütün elemanları ölen grup oyunun mağlubu sayılır.

RPG, sıra tabanlı bir tahta oyunu olmasına rağmen bilinemeyecek unsurlar içerdiğinden MiniMax algoritması kullanılmamıştır. Bunun yerine her el sonunda, hangi düşmana saldırılacağının ortaya konulduğu bir değerlendirme fonksiyonu kullanılmıştır.

Elinizde bulunan "Yapay Zekâ Yöntemleriyle Oyun Geliştirme" adlı tez çalışmasında, öncelikle yapay zekâda kullanılan başlıca yöntemler sıralanmıştır. Bu yöntemlerden MiniMax; sıfır toplamlı, tam bilgili ve sıra tabanlı bir oyun olan Tic Tac Toe'da kullanılmıştır. İkinci bir oyun olan RPG'de ise bir ağırlıklı toplamlar yöntemi kullanılmış ve ağırlıkların optimizasyonu için bir hata güdümlü eğitim yöntemi önerilmiştir.

Bu çalışmanın ikinci bölümünde Yapay Zekâ'da kullanılan başlıca yöntemleri, üçüncü bölümünde yazar tarafından geliştirilmiş ve yukarıda belirtilmiş olan iki bilgisayar oyunu ve Yapay Zekâ tasarımlarını, dördüncü bölüm olan "Bulgular ve Tartışma" bölümünde ortaya konulmuş olan Yapay Zekâ yöntemlerinin başarımlarını ve son bölüm olan "Sonuçlar ve Yorum" bölümünde bu çalışmanın genel başarımını ve ileride bu tezi başvuru amaçlı olarak kullanacak araştırmacılara önerileri bulacaksınız.

2 YAPAY ZEKÂ YÖNTEMLERİ

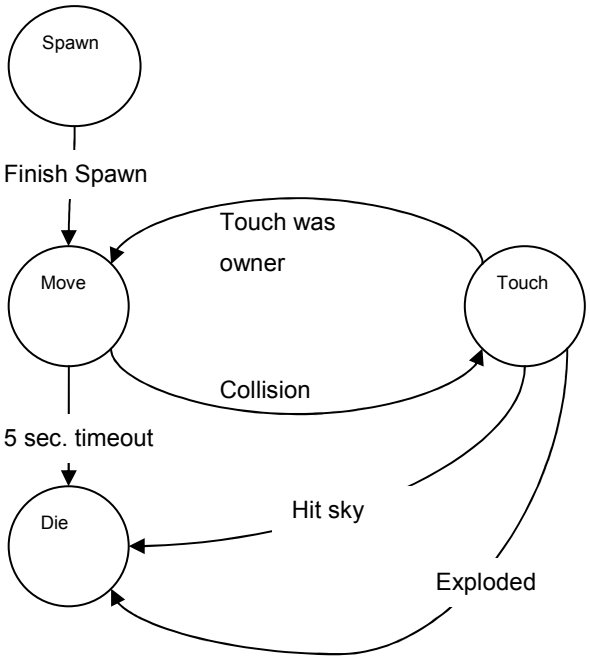
2.1 Yapay Zekâ'da Kullanılan Karar Mekanizmaları

2.1.1 Sonlu durum makineleri

Sonlu durum makineleri, sonlu sayıda durumların bulunduğu ve bu durumlar arasındaki geçişlerin belirli kurallara bağlandığı bir karar verme yöntemidir. Sonlu durum makineleri temel olarak dört bileşenden oluşur:

- Durum
- Başlangıç Durumu
- Alfabe
- Geçiş Fonksiyonu

Sonlu durum makinelerinde, durumlar arası geçiş, tanım uzayı alfabe olan geçiş fonksiyonunun sonucuna göre yapılır. Id Software'in efsanevi oyunlarından Quake oyunundaki roket mermisinin durum çizelgesi Şekil 2.1'de verilmiştir:



Şekil 2.1: Örnek Sonlu Durum Makinesi [4]

Sonlu durum makineleri genellikle karar vermekten çok karar kalıbı belirlemeye yönelik bir yöntem olarak kullanılır. Örneğin FPS tarzı bir oyunda bilgisayar, sağlık durumunun kötü gittiğini düşünüyorsa geri çekilme durumuna girer. Bu durumda alacağı kararların oluşturduğu uzay diğer durumlardakinden farklı olur.

2.1.2 Öznel beklenen yarar (SEU)

Öznel beklenen yarar, 1954 yılında Leonard Jimmie Savage tarafından ortaya atılmış bir karar kuramı yöntemidir. Bayes olasılığı kuramına dayanan bir öznel olasılık analizi ile öznel bir yarar fonksiyonunu bir araya getirir.

Savage ispatlamıştır ki [5] bir kesin olmayan olayın olası sonuçları $\{x_i\}$ ise ve her birinin kişiye olan yararı (utility) $u(x_i)$, ve her birinin meydana gelme olasılığı $P(x_i)$ ise kişinin Öznel Beklenen Yararı (SEU) Denklem 2.1'de verilmiştir.

$$SEU = \sum_i u(x_i) \cdot P(x_i) \tag{2.1}$$

Alınan bir karar sonuçları $\{y_i\}$'ye getirir ve bu sefer beklenen yarar Denklem 2.2'deki gibi olur.

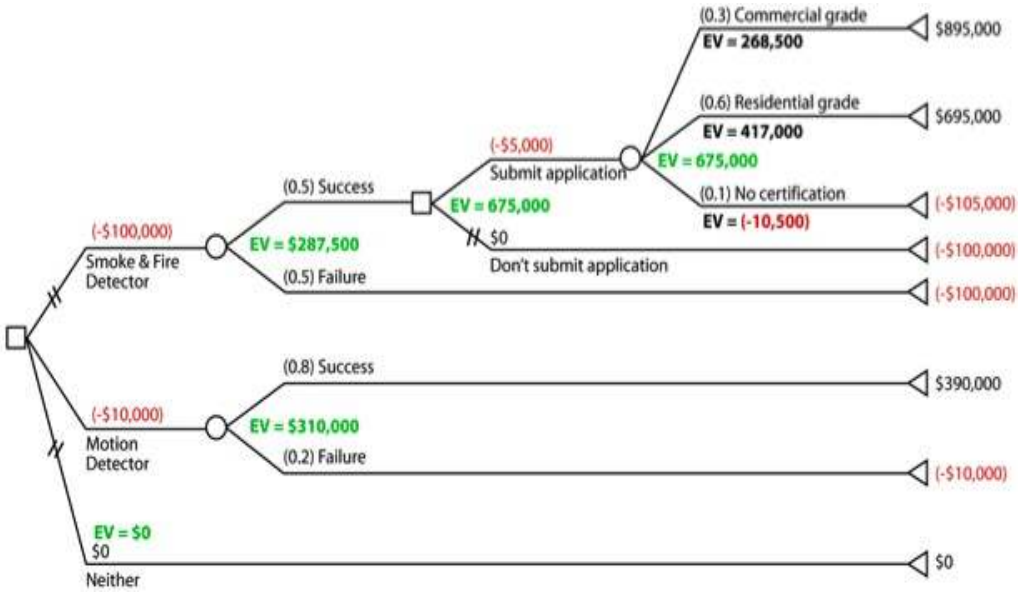
$$SEU = \sum_i u(y_i) \cdot P(y_i) \tag{2.2}$$

SEU yöntemi, olası bütün sonuçların, sonuçların olasılıklarının ve öznel olan yarar fonksiyonlarının kusursuz şekilde bilindiğini kabul eder ve bu yüzden hesaplanabilme yönünden eksik kalır. Yine de mikro-uzay problemlerine uygulanabilirliği yüksektir.

SEU yöntemi, bir ajanın mantıklı bir şekilde davranmasının ne anlama geldiğini ilk kez aksiyomlanabilmiş bir yolla anlatılabildiği ilk kuram olma özelliği nedeniyle, karar verme konusunda önemli bir kuramdır.

2.1.3 Karar ağaçları

Karar Ağaçları, stratejik kararların verilmesinde yardımcı olarak kullanılan birer çizelgedir. Çizelge bir ağaca benzer ve ağacın her dalı o an alınabilecek bir kararı simgeler. Alınan her bir kararın getirisi ve götürüsü muhasebe edilir ve başarı yüzdesi ile çarpılır. Bu hesaplamadan en yüksek puanla çıkan olasılık en kârlı olasılık olarak değerlendirilebilir. Şekil 2.2, örnek bir karar ağacını resmetmektedir.



Şekil 2.2: Örnek Karar Ağacı [6]

Örnek şirketin senaryosuna göre, ellerindeki iki projeden en çok birini yapabilecek olan şirketin hareket algılayıcı ya da duman ve yangın algılayıcısı projelerinden birisini seçmesi gerekmektedir. Yangın algılayıcısı projesinin başlangıç maliyeti 100.000$'dır ve başarı olasılığı %50'dir. Eğer şirket projeyi geliştirmekte başarılı olursa projeyi sektördeki diğer bütün ürünler gibi bir standartlara uygunluk testinden geçirecektir. Testin maliyeti 5.000$'dır ve eğer ticari uygunluk alırsa (olasılık %30) projenin getirisi 1.000.000$ olacaktır. %60 olan diğer bir olasılıkta yalnızca evler için bir uygunluk alınmakta ve ticari uygunluğa göre daha az olan 800.000$ getiri sağlamaktadır. Proje hiç bir uygunluk alamazsa (%10 olasılık) projenin yatırım masrafları çöpe gidecektir.

Hareket algılayıcısının ise başlangıç maliyeti 10.000$ ve başarı olasılığı %80'dir. Projede başarı sağlanırsa elde edilecek kazanç 400.000$'dır.

Karar ağaçları doldurulurken, SEU yöntemindeki gibi, bütün olasılıkların bilinmesi gereklidir. Ağacın her bir düğümünün beklenen değer (EV) formülü Denklem 2.3'te verilmiştir.

$$EV_d = \sum_i \left(Gelir_i - Gider_i\right) \cdot P(i) \tag{2.3}$$

Dikkat edilirse, karar ağaçlarının aslında yarar fonksiyonu düğümdeki kâr olan bir SEU uygulaması olduğu söylenebilir.

Bilgisayar da bir oyunda bir karar vermek durumunda kaldığında aynı şekilde bir karar ağacı yaratıp, bu karar ağacındaki hesaplamalara göre EV değeri en yüksek yolu seçmesi sağlanabilir. Ancak özellikle hesaplamalardan çok reflekslerin etkili olduğu hızlı oyunlarda (örneğin FPS tarzı oyunlarda) karar ağaçları çok daha basit olabilir, hatta Veri Madenciliğinde kullanıldığı üzere eğer-ise kurallarından oluşabilir.

2.1.4 MiniMax

Bilgisayar oyunlarında Yapay Zekâ uygulamalarında kullanılan algoritmalardan birisi de karar kuramından gelen MiniMax yöntemidir. Algoritma, temelinde sıra tabanlı, tam bilgili, sıfır toplam oyunlarda kullanılmak üzere tasarlanmıştır; ancak diğer türlerde kullanılmak üzere değiştirilebilir. Yöntem, olası en yüksek zararın en aza indirgenmesi olarak tanımlanabilir.

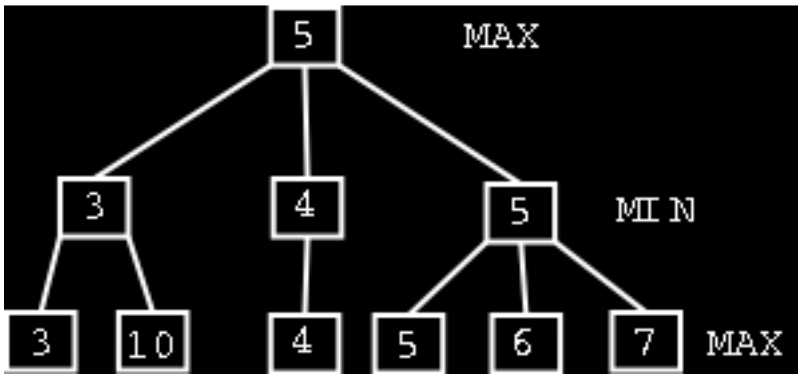
δ, θ parametresini kestiren fonksiyon ve R(θ, δ), risk fonksiyonu (genellikle kayıp fonksiyonunun integrali olarak alınır) olmak üzere;

$$\sup_{\theta} R(\theta, \tilde{\delta}) = \inf_{\delta} \sup_{\theta} R(\theta, \delta) \qquad (2.4)$$

Olduğu noktadaki R fonksiyonu MiniMax fonksiyonudur.

MiniMax fonksiyonu, temelinde sıra tabanlı, tam bilgili, sıfır toplam oyunlarda kullanılır; ancak diğer türlerde kullanılmak üzere değişikliklerde bulunulabilir.

Oyunda Min ve Max adında iki oyuncu vardır ve bilgisayar, bu iki oyuncunun da oynayabileceği bütün olasılıkları çıkartarak kendisi için en iyi oyun durumunu elde etmeye çalışır. Algoritma, bir arama ağacına dayanır (Şekil 2.3) . Arama ağacının her düğümü oyunun bir durumunu (ya da kısaca bir oyunu) tutar. Bir düğümün alt düğümleri ise o durumdan sonraki olası durumları tutar. Min ve Max ile işaretlenmiş seviyeler o seviyede oynayacak olan oyuncuyu gösterir.



Şekil 2.3: MiniMax Arama Ağacı [7]

Ağaç; derinlemesine (depth-first), o anki oyun durumundan başlayarak oyunun son durumuna kadar hesaplanır ve oyunu sonlandıran bir hamle bulunduğunda oyun sonucu Max‘e göre yorumlanır. Daha sonra ağacın dallarındaki düğümler aşağıdan yukarıya doğru, hesaplanan değerlerle doldurulur. Max'in oynadığı durumları gösteren düğümler kendi çocuklarının en büyük değerini alırken Min'in oynadığı durumları gösteren düğümler çocuklarının en düşük değerini alır.

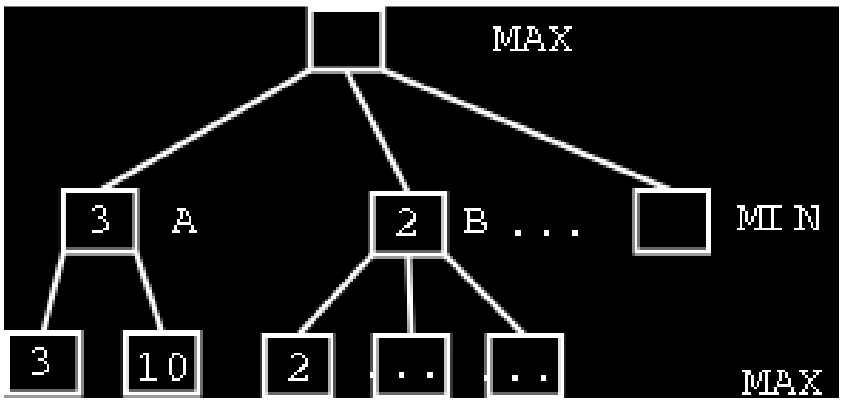
Sonuç olarak, arama ağacındaki değerler o durumun Max için ne kadar iyi bir oyun sonucu olduğunu gösterir. Max, bir hamle yaparken bu düğümlerden en büyük değerlisini seçmek isteyecektir. Buna karşılık, Min de Max'in durumunu kötüleştirmeye (yani kendi durumunu iyileştirmeye) çalışacak bir hamle yapacağından bu Max'in hamle seçimini zorlaştıracaktır.

MiniMax yöntemini kullanan bir algoritma EK-A’daki sözde-kod ile gerçekleştirilebilir.

2.1.5 Alfa / beta kesintileri

MiniMax yöntemi, oyunda olası bütün hamlelerin bellekte bir ağaç yapısında tutulması ve aramanın bütün ağaç üzerinden yapıldığı için özellikle satranç gibi hamle uzayı büyük oyunlarda işlem yükünü arttırdığı için Alfa/Beta kesintileri tekniği kullanılır.

Alfa-Beta kesintileri Şekil 2.4'e benzer ağaçlarda kullanılabilecek bir yöntemdir:



Şekil 2.4: Alfa-Beta Kesintisine Uygun Bir Arama Ağacı [7]

A ve B düğümleri MIN sırasında olduğundan A'nın değeri olan 3'ten küçük bir sayı B'nin değeri olarak seçilecek olursa B'nin bir sonraki adımda A'yı geçemeyeceği kesindir ve B düğümüyle daha fazla zaman kaybetmeye gerek yoktur.

B'nin ilk çocuğunun değeri 2 olduğu bilindiği anda diğer çocukların değerleri 2'den büyük olması koşulunda en küçük değer kalacak olan 2'nin B'nin değeri olacağı, diğer çocukların değerlerinin 2'den küçük olması koşulunda ise B'nin değerinin 2'den küçük (ve dolayısıyla MAX sırasında A'nın değeri olan 3'ü geçemeyecek) bir değer alacağından B ile ilgilenmeye gerek yoktur.

Ya da kısaca;

alfa = bilinen en iyi MAX değeri ve beta = bilinen en iyi MIN değeri olmak üzere,

1. MAX düğümlerinde, herhangi bir yolu izlemeye başlamadan önce, bir önceki yolun değerini beta değeri ile karşılaştır. Eğer değer beta'dan büyükse bu düğümü atla

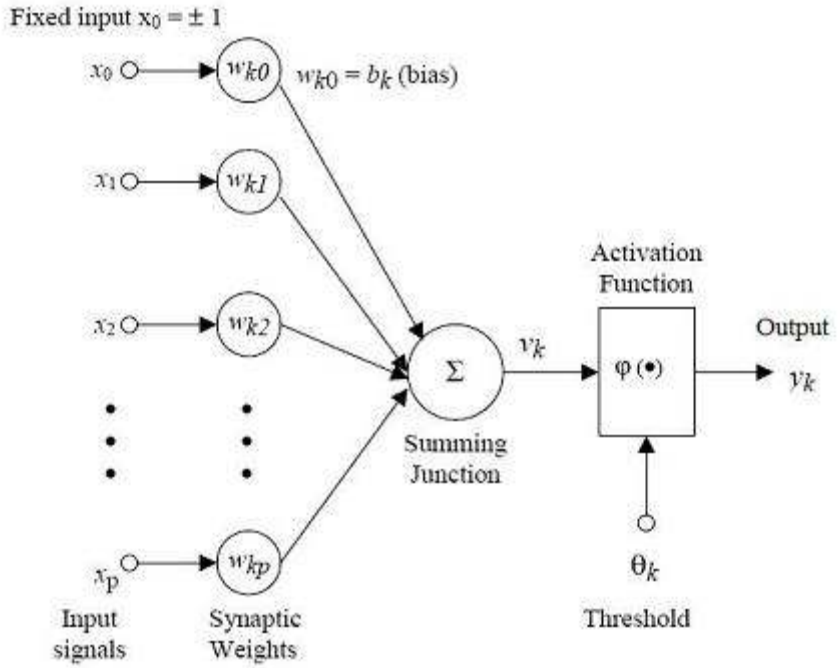
2. MIN düğümlerinde, herhangi bir yolu izlemeye başlamadan önce, bir önceki yolun değerini alfa değeri ile karşılaştır. Eğer değer alfa'dan küçükse bu düğümü atla

Alfa-Beta kesintileri, MiniMax algoritmasında yapay zekâ kalitesini düşürmeden önemli hız kazancı doğurabilir. Ancak bu hızlanmanın ölçeği arama ağacının yapısına bağlıdır. MAX düğümlerinin değerleri küçükten büyüğe doğru sıralı ise, ya da MIN düğümlerinin değerleri büyükten küçüğe sıralı gelmişse alfa-beta kesintilerinin performansa bir katkısı bulunmaz.

2.1.6 Yapay sinir ağları

Yapay Sinir Ağları, omurgalı hayvanların sinir sistemlerinden esinlenerek ortaya konulan iteratif, öğrenmeye dayalı bir kestirim yöntemidir. Birbirleri ile iyi iletişimleri bulunan sinir hücreleri (düğümler) kendisinden önceki düğümden gelen veriyi basit işlemler (ağırlık çarpanı ve düğüm ağırlığı toplamsalı kullanarak) yaparak bir sonraki düğüme iletir. En son düğümde elde edilen sonuç, eğitim verisi ile karşılaştırılır, eğer sonucun iyileştirilmesi gerekiyorsa YSA parametreleri güncellenir.

Bir düğümün çıktısı, kendisine gelen girdilere ve her bir girdiye atanmış olan ağırlıklara bağlıdır. Bir yapay sinir ağı düğümünün matematiksel modeli Şekil 2.5 ve Denklem 2.4 ile Denklem 2.5'te gösterilmiştir.



Şekil 2.5: Yapay Sinir Ağı Düğümü [8]

$$v_k = \sum_{j=1}^{p} w_{kj} \cdot x_j \tag{2.4}$$

$$y_k = \varphi(v_k) \tag{2.5}$$

Etkinleştirme fonksiyonu, önceden belirlenmiş bir fonksiyondur ve amacı sinir çıkışını belirli bir aralıkta (genellikle [-1, 1]) arasında tutmaktır. En yaygın doğrusal olmayan etkinleştirme fonksiyonları sigmoid ve hiperbolik tanjant fonksiyonlarıdır. Denklem 2.6 sigmoid fonksiyonu, Denklem 2.7 ise hiperbolik tanjant fonksiyonunun tanımını vermektedir.

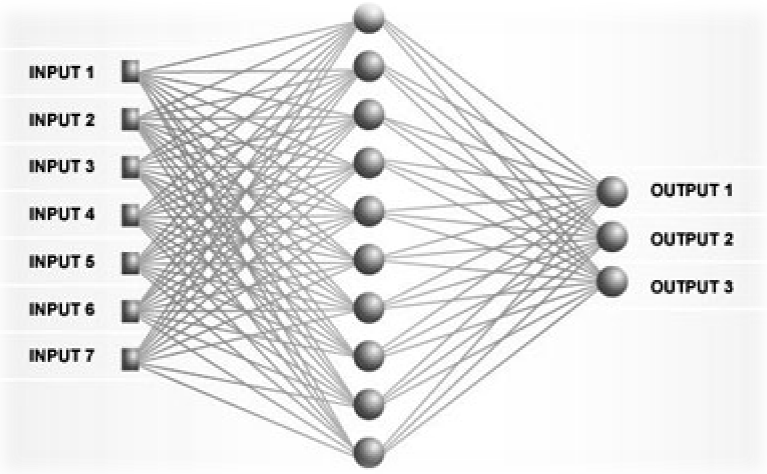
$$\varphi(v_k) = \frac{1}{1+e^{-c \cdot v_k}} \tag{2.6}$$

$$\varphi(v_k) = \tanh\left(\frac{v_k}{2}\right) = \frac{1-e^{-v_k}}{1+e^{-v_k}} \tag{2.7}$$

2.1.6.1 Yapay sinir ağı yapıları

Yapay sinir ağlarının sınıflandırılması sinir ağının yapısına (topolojisine) göre yapılır. Sık kullanılan topolojiler ileri beslemeli ağlar (feed forward networks) ve devirli ağlar (recurrent networks) olarak ayrılabilir.

İleri beslemeli ağ topolojisi, bütün sinirleri yalnızca bir sonraki ağ katmanına bağlanan ve veri akışı yalnızca ileriye doğru giden yapay sinir ağı topoloji türüdür. Türev işlemlerinin daha kolay hesaplanmasını sağladığı için diğer topolojilere göre daha çok kullanılır ve yapay sinir ağı denildiğinde genellikle akla ilk gelen topolojidir. Şekil 2.6, örnek bir ileri besleme ağı topolojisini gösterir.



Şekil 2.6: İleri Besleme Yapay Sinir Ağı [9]

Eğer ağın sinirleri arasında aynı katmanda ya da bulunduğu katmanlardan daha geride bulunan sinirlere bağları olanlar varsa ağ bir devirli yapay sinir ağıdır.

2.1.6.2 Yapay sinir ağlarında eğitim

Yapay sinir ağlarının eğitiminde gözetmenli, gözetmensiz ya da desteklenmiş eğitim yöntemleri kullanılır. Gözetmenli yapay sinir ağları, geri yayılım (back-propagation) kullanılarak yapay sinir ağının çıktılarının toplam hatasını azaltmaya çalışır. Bunun için ilk yapılması gereken şey bir hata fonksiyonu seçmektir. Hata fonksiyonu olarak karesel ortalama fonksiyonu uygundur. Denklem 2.8 toplam n adet eğitim çifti içeren eğitim kümesi ile eğitilen, çıkış vektörü o, hedef vektörü t olan bir yapay sinir ağının karesel ortalama hatasını ortaya koymaktadır.

$$E = \sum_{i=1}^{n} \|o_i - t_i\|^2 \tag{2.8}$$

Yapay sinir ağlarının eğitim problemi, bu hata değerinin minimize edilmesidir. Hücre çıkışlarını belirleyen etkinleştirme fonksiyonları sürekli ve türevlenebilir fonksiyonlardan seçildiği, hücre çıkışlarının kendi girişlerine ait ağırlıklarına bağlı olduğundan ve eğitimde değiştirilebilinecek tek parametre hücre girişlerinin ağırlıkları olduğundan E gradyan indirgeme (gradient descent) yöntemi ile sıfıra yakınsayabilir [10]. Denklem 2.9a gradyan indirgeme yöntemi ile toplam l adet ağırlık değeri bulunan bir yapay sinir ağındaki E hatasını mimimize etmek için ilgili ağırlık değerine eklenmesi gereken farkı göstermektedir. Denklemdeki γ değeri önceden belirlenmiş bir eğitim sabitidir. [11]

$$\nabla E = \left(\frac{\delta E}{\delta w_1}, \frac{\partial E}{\partial w_2}, \cdots, \frac{\partial E}{\partial w_l}\right) \tag{2.9a}$$

$$\Delta w_i = -\gamma \cdot \frac{\partial E}{\partial w_i} \tag{2.9b}$$

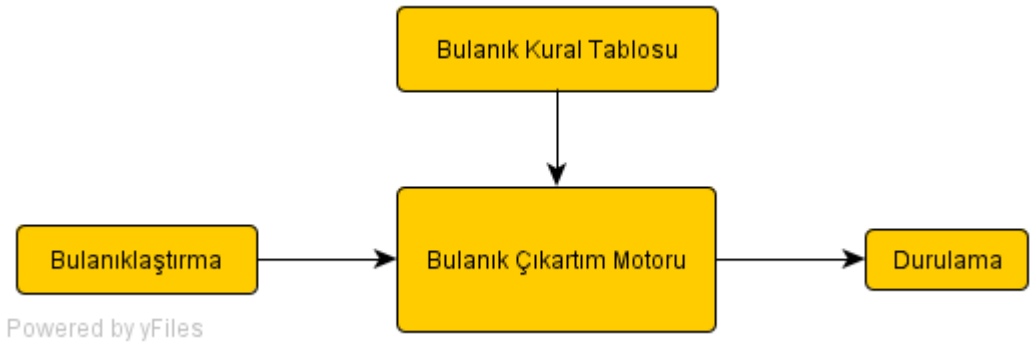
$$w_i(t+1) = w_i(t) + \Delta w_i \tag{2.9c}$$

Gözetmensiz eğitimde yapay sinir ağına giriş çıkış çiftleri verilmez. Onun yerine sinir ağı verilen girişlere çıktılar üretir ve çıktının hatasını kendisi tahmin etmeye çalışabilir [12]. Gözetmensiz eğitimin sık kullanıldığı amaçlar arasında sınıflandırma, istatistiksel dağılım çözümleri, sıkıştırma ve süzme (filtreleme) bulunur.

Desteklenmiş eğitim, hem gözetmenli eğitimi hem de gözetmensiz eğitimi andırır. Desteklenmiş eğitimde yapay sinir ağına girdi-çıktı çiftleri verilmez, ancak sinir ağının verdiği çıktıya göre bir “ödül” ya da “ceza” verilerek arzulanan hedefe yaklaşması sağlanır. Yapay sinir ağı, ödülü arttırmak ya da cezayı azaltmak için hücreler arası ağırlıklarda güncellemelerde bulunur.

2.1.7 Bulanık mantık

Bulanık mantık sistemleri, çözülmesi istenen problemlerin uzmanlarına ya da bilgi bankalarına başvurularak oluşturulan eğer-ise (if-then) kurallarının sürekli üyelik fonksiyonları ve bu fonksiyonlarla ilişkilendirilen dilsel sözcüklere dayanır. Şekil 2.7 bir Tsukamoto bulanık sistemin bileşenlerini göstermektedir.



Şekil 2.7: Tsukamoto Modeli Bulanık Sistem Çizelgesi

Bir başka bulanık model olan Sugeno modelinde ise durulama olmaz; çünkü Bulanık Çıkartım Motorunun çıkış değeri duru değerlerdir.

2.1.7.1 Bulanık kural tablosu

Bulanık kural tablosu bulanık çıkartım motorunun çalışmasını belirleyen eğer-ise kurallarından oluşan bir tablodur. Tablodaki kurallarda bulunan “eğer” ifadeleri, dilsel sözcükler barındırır. Tablo 2.1 olası bir Tsukamoto modeli araç durdurma sisteminin olası bulanık kural tablosunun bir bölümünü ortaya koymaktadır.

Tablo 2.1: Örnek Tsukamoto Bulanık Kural Tablosu

HIZ	HIZLANMA	UYGULANACAK FREN MİKTARI
Çok düşük	Sıfır	Çok az
Düşük	Sıfır	Az
Orta	Sıfır	Orta

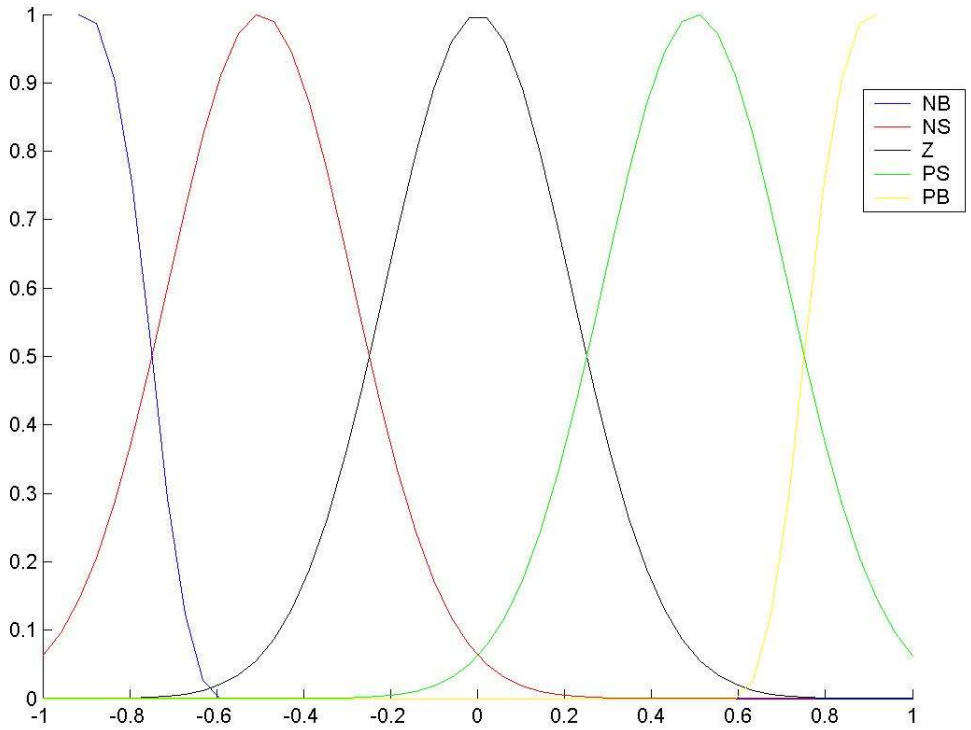
Aynı sistemin Sugeno modeli kural tablosunun bir bölümü ise Tablo 2.2’de verilmiştir. Çıkış verileri bulanık değil, duru değerlerdir.

Tablo 2.2: Örnek Sugeno Bulanık Kural Tablosu

HIZ	HIZLANMA	UYGULANACAK FREN MİKTARI
Çok düşük	Sıfır	$c_1 * V+d_1$
Düşük	Sıfır	$c_2 * V+d_2$
Orta	Sıfır	$c_3 * V+d_3$

2.1.7.2 Bulanıklaştırma

Bulanıklaştırma, kesin değerleri bulanık çıkartım motorunun anlayacağı duruma getirmek anlamına gelir. Bulanıklaştırma işi, bulanık kural tablosunda belirlenen girdilerin sayısında ve tepe değerleri 1 olan sürekli üyelik fonksiyonları tanımlamaktır. En sık kullanılan üyelik fonksiyonları gauss, üçgen ve yamuk sürekli fonksiyonlarıdır. Her bir üyelik fonksiyonu giriş uzayının belirli değerleri arasına hâkimdir ve üyelik fonksiyonunun tanımlı olduğu aralığa dilsel olarak anlamlı bir ad verilir. Şekil 2.8, 5 adet üyelik fonksiyonu içeren ve bir aracın hızını bulanıklaştıran bir bulanıklaştırma örneği göstermektedir.



Şekil 2.8: Bulanık Üyelik Fonksiyonları [13]

Şekil 2.8'deki örnekte giriş değeri -0,2 olan bir araç hem *PS (positive-small)* hem de *Z (zero)* hem de *NS (negative-small)* üyeliklerine dâhildir.

2.1.7.3 Bulanık çıkartım motoru

Bulanık çıkartım motoru, sistemin çıkışını kurallar tablosuna dayanarak ortaya çıkartma işini üstlenir. Çıkışı yine bulanık bir sonuç kümesi olan bulanık çıkartım motorunun işi iki aşamadan oluşur.

İlk aşama olan toplama (aggregation), kural tablosunun "eğer" kısımlarını hesaplar. Birden fazla üyelik fonksiyonunun üyeliğine giren değerler için değerlerin minimumu (MIN çıkartım motoru), çarpımı (PROD çıkartım motoru) ya da belirlenen başka fonksiyonlar kullanılabilir.

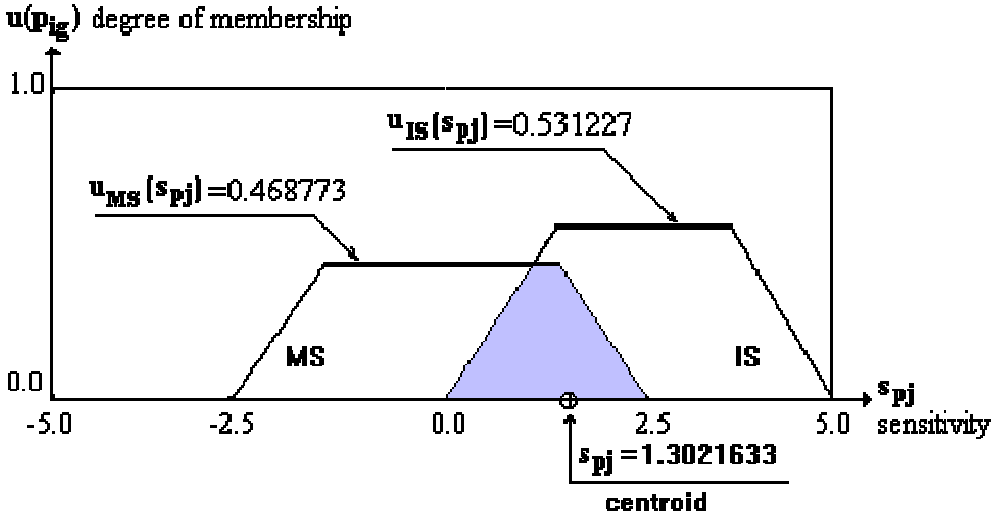
İkinci aşama olan birleştirme (composition) ise kural tablosunun "ise" kısımlarındaki değerleri hesaplar. Birden fazla üyelik fonksiyonunun üyeliğine giren çıkış değerleri

için değerlerin en büyüğü (MIN çıkartım motoru), toplamı (PROD çıkartım motoru) ya da belirlenen başka fonksiyonlar kullanılabilir.

İkinci aşamanın sonunda, durulanacak olan değerler, sisteme verilen girişlerin karşılık düştüğü çıkış üyelik fonksiyonlarının birleştirme aşamasından çıkan birleşimleridir.

2.1.7.4 Durulama

Durulama aşaması, bulanık çıkartım motorundan gelen üyelik fonksiyonları birleşiminin tekil ve net bir değer olarak ortaya koyulduğu aşamadır. Çıkartım motorundan elde edilen çıkış üyelik fonksiyonları birleşimine bir örnek Şekil 2.9'da verilmiştir.



Şekil 2.9: Bulanık Çıkış Üyelik Fonksiyonları Birleşimi [14]

Birleşim kümesinden duru ve net bir değer çıkartmanın yine birden fazla yolu vardır. Bunlardan bazıları maksimumların merkezi (CoM – center of maximum), ağırlık merkezi (CoA – center of area), maksimum ortalama (MoM – mean of maximum) yöntemleridir.

Bulanık mantık sistemleri özellikle kontrol alanında yaygınlaşan bir kullanım alanına sahiptir.

2.1.8 Bulanık sinir ağları

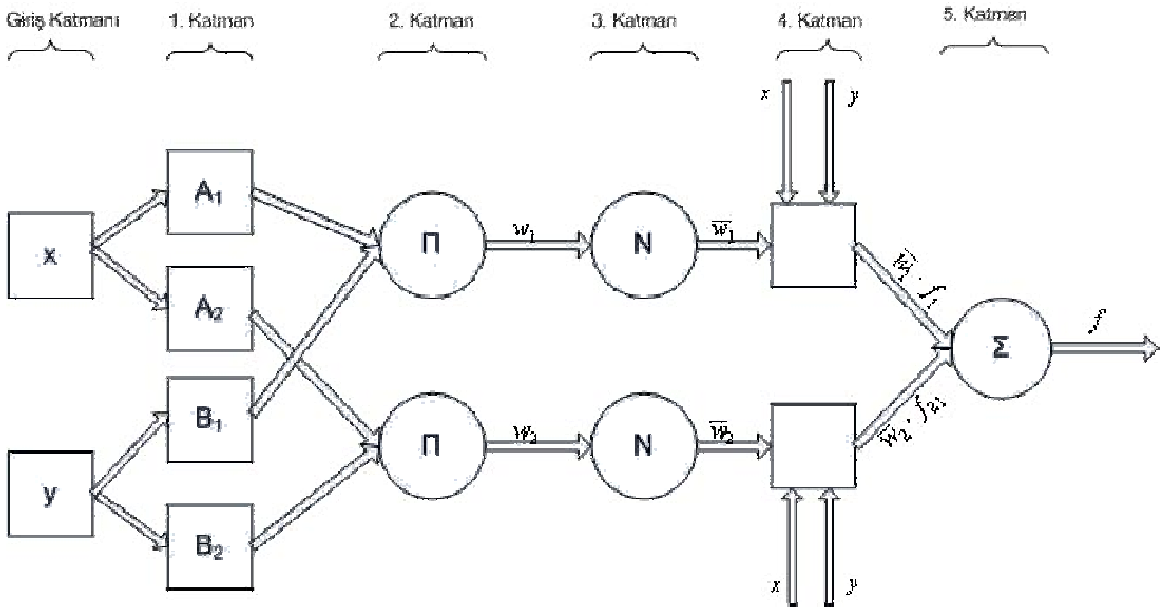
Uyarlanabilir Ağ Yapılı Bulanık Çıkartım Sistemleri (ANFIS – Adaptive Neuro-Fuzzy Inference System) ya da kısaca Bulanık Sinir Ağları, standart Bulanık Çıkartım Sistemlerinin Yapay Sinir Ağı Modeline oturtulmuş biçimidir. Sistem, başlangıçta tanımlanmış bir bulanık Sugeno ya da Tsukamoto modelinin çıkışlarını üretecek bir ağ yapısı ile tanımlanır ve bulanık modelin üyelik fonksiyon parametreleri, yapay sinir ağlarında kullanılan eğitim yöntemleri ile güncellenir. [13]

2.1.8.1 Bulanık model ve ilişkin ağ yapısının çıkarımı

Yukarda da belirtildiği gibi, ANFIS yapısı önceden tanımlanmış olan bir bulanık çıkartım modelinin çıkışını verecek bir yapay sinir ağı yapısındadır. Örnek bir iki girişli ve iki kurallı Sugeno sistemi ve ona ilişkin bulanık kurallar Tablo 2.3'te, ilişkin ANFIS sistemi Şekil 2.10'da verilmiştir:

Tablo 2.3: ANFIS Bulanık Kural Tablosu

Girişler:	x	y
Üyelik Fonksiyonları:	A_1 , A_2	B_1 , B_2
Kurallar:	Eğer x A_1 ve y B_1 ise $f_1 = p_1(x) + q_1(y)+r_1$ Eğer x A_2 ve y B_2 ise $f_2 = p_2(x) + q_2(y)+r_2$	



Şekil 2.10: ANFIS Mimarisi

2.1.8.2 Bulanık ağ yapısının katmanları ve işlevleri

ANFIS yapısında her katman bir sonraki katmana bağlıdır ve her katmanda bulanık sistemin işleyişinin farklı bir adımı gerçeklenir.

1. katmandaki hücreler giriş değerlerinin üyelik fonksiyonları çıkışlarını verir. (*x* için $A_1(x)$ ve $A_2(x)$, y için $B_1(y)$ ve $B_2(y)$)

2. katmandaki hücreler PI hücreleri olarak adlandırılır ve çıkış olarak kendisine gelen bütün sinyallerin aritmetiksel çarpımını verir. i'nci PI hücresi için;

$$w_i = A_i(x) \cdot B_i(y) \tag{2.10}$$

w_i değerleri, işlenen kuralın tetik değeri olarak adlandırılır.

3. katmandaki hücreler N hücreleri olarak adlandırılır ve normalize edilmiş tetik değerlerini üretir: i'nci N hücresi için;

$$\overline{w_i} = \frac{w_i}{\sum w} \tag{2.11}$$

4. katmandaki hücreler her bir kuralın giriş değerlerine göre çıkışını hesaplar ve 5. katmana normalize edilmiş ağırlığı ile bu fonksiyon çıkışının aritmetik çarpımını iletir. i'nci hücre için;

$$\overline{w_i} \cdot f_i = \overline{w_i} \cdot (p_i \cdot x + q_i \cdot r_i) \tag{2.12}$$

5. katman SİGMA adı verilen tek bir hücreden oluşur ve bütün 4. katman çıkışlarının aritmetiksel toplamını döndürür.

$$f = \sum \overline{w_i} \cdot f_i = \frac{\sum w_i \cdot f_i}{\sum w_i} \tag{2.13}$$

Tablo 2.3 verilmiş olan Sugeno modelini Denklemler (2.10 – 2.13) ile tamamen gerçekleyen bir ANFIS yapısı böylece tanımlanmış olur. f çıkış değerine dikkat edilecek olursa ağırlıklı ortalama yöntemi kullanılmış bir Sugeno sisteminin çıkış fonksiyonuna eşit olduğu görülebilir.

2.1.8.3 Bulanık ağlarda eğitim

Elde ettiğimiz yapının uyarlanabilirliği, bu yapının eğitimi ile mümkün olmaktadır. Örnek ANFIS yapısının çıkış denklemini bir tümevarım için kullanalım:

$$f = \frac{w_1}{w_1 + w_2} \cdot f_1 + \frac{w_2}{w_1 + w_2} \cdot f_2 \tag{2.14a}$$

$$= \overline{w_1} \cdot (p_1 x + q_1 y + r_1) + \overline{w_2} \cdot (p_2 x + q_2 y + r_2) \tag{2.14b}$$

$$= (\overline{w_1} x) p_1 + (\overline{w_1} y) q_1 + (\overline{w_1}) r_1 + (\overline{w_2} x) p_2 + (\overline{w_2} y) q_2 + (\overline{w_2}) r_2 \tag{2.14c}$$

Denklemi, f çıkışının p_i, q_i ve r_i sonuç/doğrusal (consequent) değişkenleri üzerinde doğrusal olduğunu gösterir. Bulanık Sugeno sisteminin diğer parametreleri de (üyelik fonksiyonlarına ait parametreler) koşul/doğrusal olmayan (premise) değişkenler olarak adlandırılır.

Eğitim işleyişi temel olarak iki aşamaya ayrılabilir: İlk aşama, ileri geçişte (forward pass), mevcut bulanık sistemde 4. katmana kadar olan çıkışlar hesaplanır ve bu noktada doğrusal parametreler LS yöntemi kullanılarak güncellenir. İkinci aşama, geri geçişte (backward pass), bulanık sistem çıkışının hata sinyalleri geri döndürülerek gradyan azaltımı yöntemi ile doğrusal olmayan parametreler güncellenir. Bu eğitim yöntemine karma eğitim (hybrid learning) adı verilmiştir.

2.1.8.4 İleri yayılım ve LS yöntemi

LS yöntemi genel olarak,

$$y = \theta_1 f_1(u) + \theta_2 f_2(u) + \ldots + \theta_n f_n(u) \tag{2.15}$$

Şeklinde tanımlanan sistemlerde m ≥ n adet bilinen giriş – istenen çıkış ikilisi kullanarak sistemi en az hata ile yansıtan θ bilinmeyen parametrelerini bulmak için Denklem (2.16-2.17d) tanım ve denklemleri kullanır:

$$\begin{cases} f_1(u_1)\theta_1 + f_2(u_1)\theta_2 + \ldots + f_n(u_1)\theta_n = y_1 \\ f_1(u_2)\theta_1 + f_2(u_2)\theta_2 + \ldots + f_n(u_2)\theta_n = y_2 \\ \vdots \\ f_1(u_m)\theta_1 + f_2(u_m)\theta_2 + \ldots + f_n(u_m)\theta_n = y_m \end{cases} \tag{2.16}$$

$A\theta = y$, öyle ki; (2.17a)

$$A = \begin{bmatrix} f_1(u_1) & \cdots & f_n(u_1) \\ \vdots & \ddots & \vdots \\ f_1(u_m) & \cdots & f_n(u_m) \end{bmatrix}_{m \times n} \quad (2.17b)$$

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}_{n \times 1} \quad (2.17c)$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}_{m \times 1} \quad (2.17d)$$

m > n olduğu koşullarda θ matrisinin tersi alınamayacağından tam duyarlı (sıfır hatalı) bir sonuç ortaya konamayabilir. Bunun için Denklem 2.17a'ya bir hata parametresi eklenir ve bu hatanın karesini en aza indirgeyecek bir $\hat{\theta}$ sütun vektörü hesaplanır:

$$A\theta + e = y \quad (2.18a)$$

$$e = y - A\theta \quad (2.18b)$$

$$E(\theta) = \frac{1}{2} e^T e = \frac{1}{2} (y - A\theta)^T (y - A\theta) \quad (2.18c)$$

$$\frac{\partial E}{\partial \theta} = \frac{1}{2} \left(-A^T (y - A\theta) - A^T (y - A\theta) \right) = -A^T (y - A\theta) = 0 \quad (2.18d)$$

$$-A^T y - A^T A\theta = 0 \quad (2.18e)$$

$$\hat{\theta} = (A^T A)^{-1} A^T y \quad (2.18f)$$

LS yöntemi ANFIS sistemin ileri geçişine uygulanırken bilinen parametreler olarak $\overline{w_i}$, x_i ve y_i alınırken; p_i, q_i ve r_i parametrelerinin uygun değerleri aranır. Diğer bir deyişle A matrisi $\overline{w_i}$, x_i ve y_i parametrelerini içeren m x n boyutlarında bir matris, θ de; p_i, q_i ve r_i doğrusal parametrelerini barındıran n x 1 lik bir sütun vektörüdür.

2.1.8.5 Geriye yayılım ve gradyan indirgemesiyle eğitim

Gradyan indirgemesi ile eğitim yöntemi, yapay sinir ağlarında sık kullanılan bir eğitim yöntemidir. Temelde amaç, yapay sinir ağından elde edilen çıkışın hatasının karesini hata fonksiyonunun yapay sinir ağı parametrelerine göre türevini sıfırlayarak en aza indirmektir. Yapay sinir ağlarında bu parametreler sinir hücrelerine gelen sinyallerin ağırlıkları iken ANFIS sistemlerde Denklem 2.14a'da belirtilmiş olan koşul parametrelerdir. Denklemler (2.19a-2.20), karesel hatanın ve bu hatanın gradyan indirgemesi kullanılarak geriye yayılımının hesabını vermektedir.

$$e = y_{set} - f \tag{2.19a}$$

$$E = \frac{1}{2}e^2 \tag{2.19b}$$

$$\alpha_{k+1} = \alpha_k - \eta \frac{\partial E}{\partial \alpha_k} \tag{2.20}$$

Denklem 2.20'de, α sayıları ANFIS sistemin koşul değişkenlerinin her birini temsil etmektedir. Denklem 2.14a'ya dayanarak Denklem 2.20'deki türevin w_i değerlerine bağlı olduğu söylenebilir. Bir adım daha ileriye gidilirse Denklem 2.10'un da w_i değerlerinin de girişlerin üyelik fonksiyonlarına bağlı olduğu görülür. Açıktır ki, üyelik fonksiyonları da kendi parametrelerine bağlıdır. Sonuç olarak Denklem 2.20, her bir üyelik fonksiyonu üzerinde işletildiğinde bir sonraki eğitim adımında (epoch) hatayı sıfıra biraz daha yaklaştıracak bir değere ulaşacaktır.

2.1.9 Genetik algoritmalar

Genetik algoritmalar, arama ve optimizasyon barındıran birer uyarlanabilir stokastik optimizasyon algoritmasıdır. Genetik algoritmalar ilk kez 1975 yılında Holland tarafından kullanılmıştır.

Temel fikir, doğal seçilimin basit bir örneğini işleterek verilen çözümler arasından en iyi olanı seçmektir. İlk aşama örnek çözümlerin mutasyonu ya da rastgele değiştirilmesinden oluşur. İkinci adım bir seçilim adımıdır ve genellikle doğal seçilimi öykünen bir uygunluk fonksiyonunun değerlendirmesi eşliğinde yapılır. Bu iki adım, en uygun çözüm bulununcaya kadar yinelenir [15].

Genetik algoritmalar, pek çok farklı uygunluk fonksiyonu ve çaprazlama yöntemi kullansa da temel algoritma Ek – B'deki gibidir.

2.2 Eğitim yöntemleri

Öğrenme terimi psikolojide, bir varlığın davranışlarının verilen bir durumda ya da verilen durumla birlikte yinelenen deneyimlerine bağlı olarak değişmesi olarak tanımlanır. Yapay Zekâda, makine öğrenimi (ya da eğitimi) bir Yapay Zekâ sisteminin başarımını zaman içinde arttırması olarak tanımlanabilir [16].

2.2.1 Hata güdümlü öğrenim

Hata güdümlü öğrenim, makinenin bir sorunu çözerken önce hatalar yapmasını ve sonraki adımlarda bu hataları yinelememesini sağlamaktır. Bu öğrenim yöntemi, insanların öğrenim yoluyla benzerlikler gösterir. Bir insan nasıl bir kere yaptığı bir hatadan bir şeyler öğreniyor ve aynı hatayı ya da benzer hataları aynı ya da benzer sorunları çözerken yapmıyorsa bir makine de benzer şekilde programlanabilir.

İlk bakışta yapay sinir ağlarındaki "Gözetmenli Eğitim"e benzer gibi görünse de aslında "Gözetmensiz Eğitim"e benzemektedir; çünkü gözetmenli eğitimdeki gibi dışarıdan alınan eğitim verileri yoktur. Sistem kendi hata fonksiyonlarını kendisi çıkartmak durumundadır.

2.2.2 Eğitici tarafından öğrenim

Eğirici tarafından öğrenim, makinenin hangi durumda nasıl davranması gerektiğinin işin bir uzmanı tarafından makineye aktarıldığı öğrenim çeşididir. Örneğin bir bilinmeyenli bir denklemi çözerken öğretici, makineye “bilinmeyenleri eşitliğin soluna, bilinenleri sağına taşı” diyerek makineye sorunu çözebilmesi için yardım edebilir. Buradaki temel sorun eğiticinin makineyle anlaşma yoludur.

2.2.3 Keşifle öğrenim

Keşifle öğrenim diğer öğrenim yöntemlerinden biraz daha farklıdır. Öğrenimin amacı bir hedefe ulaşmak değil, yalnızca daha fazla bilgi sahibi olmak ve veri tabanındaki kavram zenginliğini arttırmaktır. Makinenin yaptığı tek şey yeni bir şeyler öğrenebileceği ilgi çekici bilgilerin bulunduğu kaynakları aramaktır. Eğitim sonu da keşfedilecek hiçbir şeyin kalmadığı nokta değil, verilmiş görevlerle ilgili yeterince bilgi sahibi olunduğu noktadır.

Makine, verilen görevleri bir “ilgi çekicilik” sıralamasına koyar ve bazı görevlerdeki bazı bilgilere gerekli ilgi çekiciliğin altında kaldığından bakmaz. İlgi çekicilik de yine bir matematiksel fonksiyon olarak belirtilir ve keşifle öğrenmenin en büyük sorunlarından biridir; çünkü çok iyi seçilmemiş bir ilgi çekicilik fonksiyonu makinenin çok gerekli bilgileri görmezden gelmesine neden olabilir.

3 YAPAY ZEKÂ ÖRNEK UYGULAMALARI

3.1 Geliştirme Ortamı

Tic Tac Toe ve RPG oyun yazılımları Windows ortamında Microsoft'un .NET Framework 2.0 kitaplıkları kullanılarak C# dili ile yazılmıştır. Yazılımın çalıştırılabilmesi için Microsoft .NET Framework 2.0'ın bilgisayara kurulması gerekmektedir. Yazılım Mono kitaplıkları üzerinde denenmemiştir.

3.2 Geliştirilen Oyunlar

3.2.1 Tic tac toe – bir zekâ oyunu

3.2.1.1 Oyunun ve kuralların tanımlanması

Tic Tac Toe, 3x3'lük bir tahtanın boş olan bir karesine, iki oyuncunun sırayla kendi işaretlerini koymalarından oluşur. Dikey, yatay ya da çapraz üçlüyü tamamlayan oyuncu oyunu kazanır. Eğer toplam 9 hamle sonunda hiç bir oyuncu bir üçlü yapamazsa oyun berabere sonuçlanır.

3.2.1.2 Kullanılan yapay zekâ modeli

Tic Tac Toe,

- Sıra tabanlı,
- Tam bilgili,
- ve sıfır toplam bir oyun olduğu için makinelere MiniMax yöntemiyle oynatılabilir.

Tic Tac Toe, arama ağacında çok fazla düğüm içermeyeceğinden (toplam 9! = 362880) MiniMax algoritması yalın olarak kullanılmıştır. Eğer oyun tahtası büyütülmek istenirse yalın MiniMax çok yavaş kalabilir, zira 5x5'lik bir tahta için toplam düğüm sayısı $15{,}5*10^{24}$ olur. Büyük ağaçlar makineler için hem saklamada hem aramada daha zor olurlar, bu yüzden alfa-beta kesintileri gibi yöntemlerle ağaç aramaları sınırlandırılmalıdır.

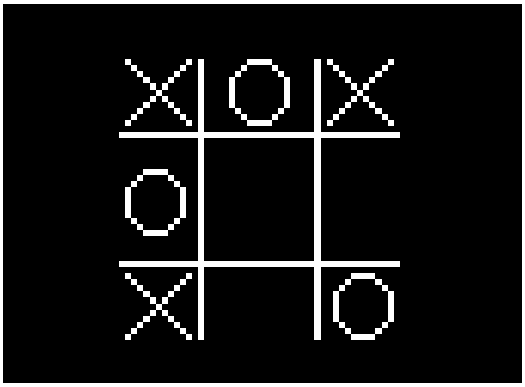
Tic Tac Toe'da kullanılan oyun sonu değerlendirme fonksiyonu denklem 3.1'de verilmiştir.

$$e = sgn(player) \cdot \big((\max(pieceCountOnALine) \cdot 100) + (9 - nodeLevel)\big) \qquad (3.1)$$

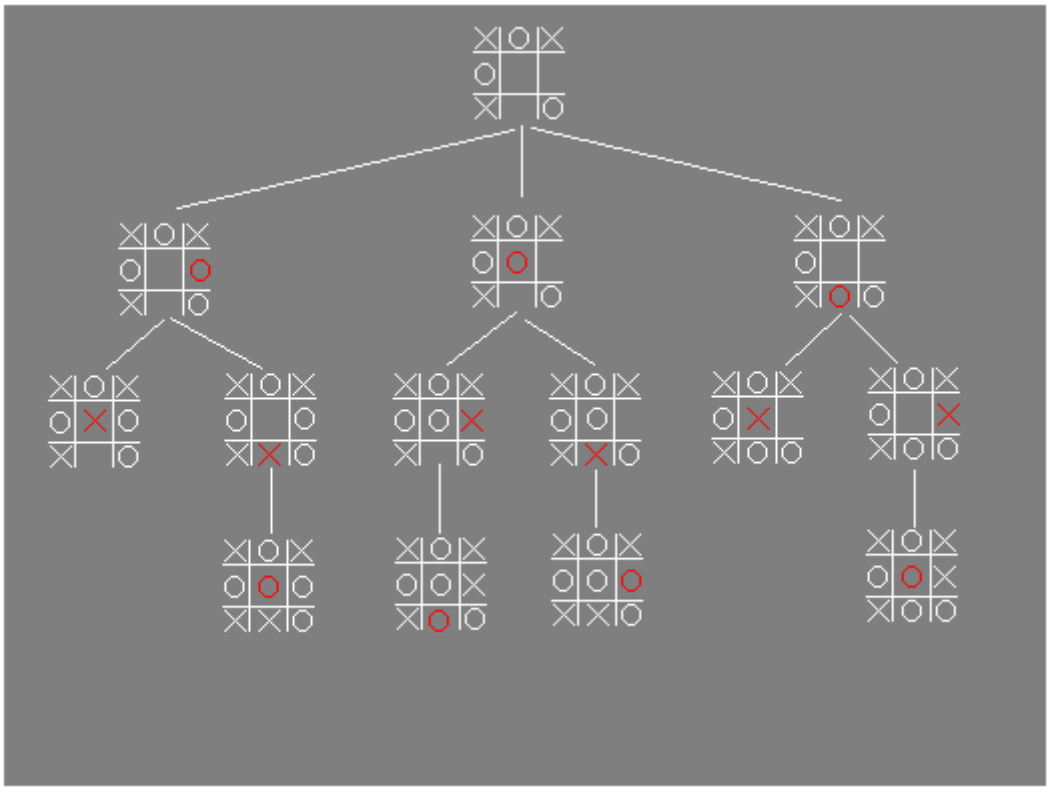
Şekil 3.1'deki durumda bilgisayar önce Şekil 3.2'deki arama ağacını çıkartır. Kırmızı ile işaretli alanlar o seferde yapılan hamleyi gösterir. Başlangıç koşulları,

- İlk başlayan oyuncu bilgisayardır
- Bilgisayarın işareti çemberdir

olarak belirlenmiştir.

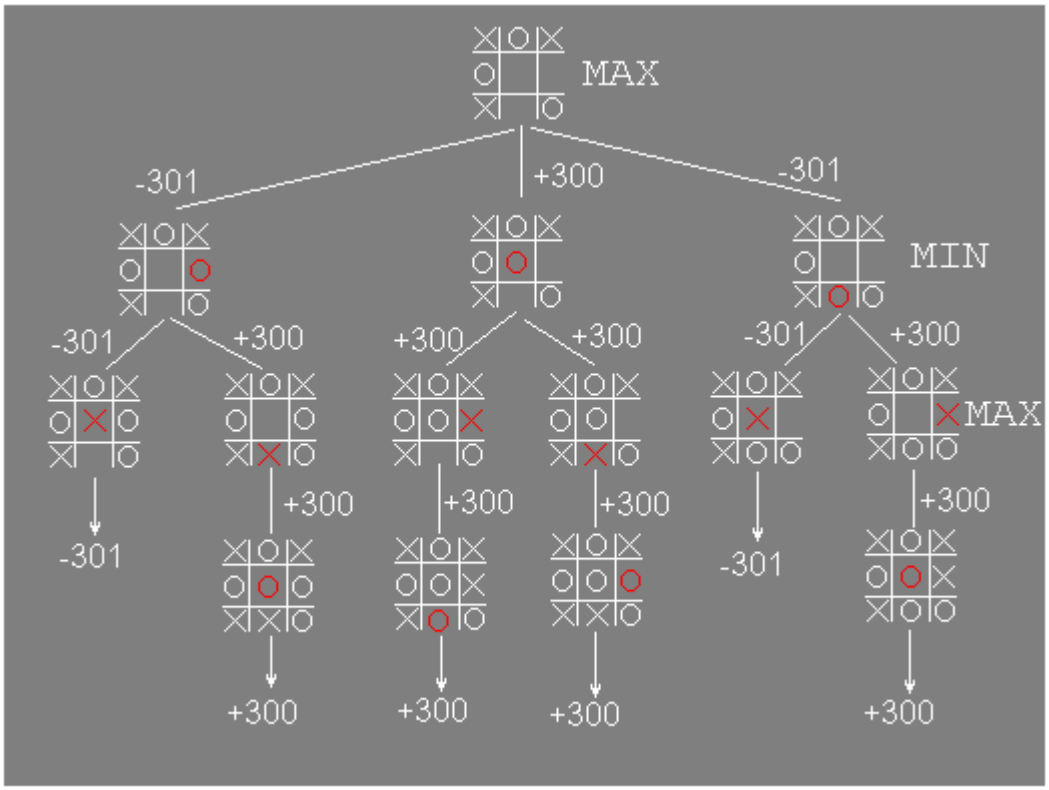


Şekil 3.1: Örnek Bir Tic Tac Toe Oyun Durumu



Şekil 3.2: Tic tac toe arama ağacı

Arama ağacında ilerlerken oyun sonunu getiren bir hamle bulunduğunda yukarıdaki fonksiyon kullanılarak oyun sonu değerlendirmesi yapılır. Her bir ağaç seviyesinin MAX ve MIN olduklarına bakarak ağaç, Şekil 3.3'te olduğu gibi aşağıdan yukarıya doğru doldurulur:



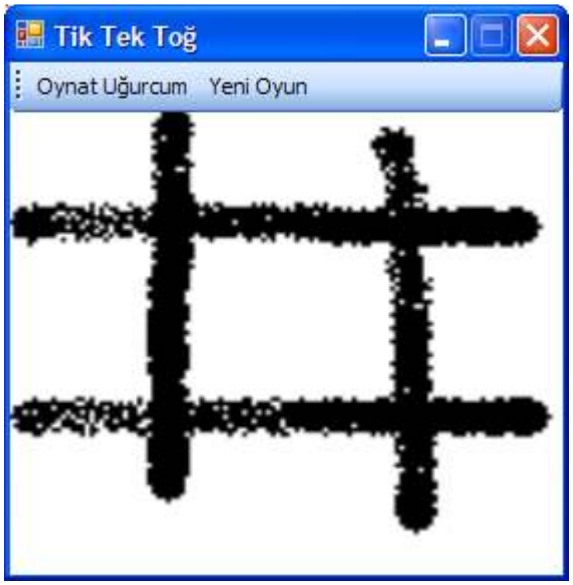
Şekil 3.3: Arama Ağacının Doldurulması

Arama ağacında, MAX düğümleri kendi çocuklarının en büyük değerlerini alırken, MIN düğümleri kendi çocuklarının en küçük değerlerini alır. Doğru parçaları yanlarındaki her sayı, o doğru parçası ile bağlanan düğümün (doğru parçasının hemen altındaki düğümün) değeridir. Yukarıdaki durumda, bilgisayar MAX oyuncusu olduğundan max(-301,300,-301) fonksiyonunu işletip, 300 sonucunu veren ortadaki düğümü seçecektir; yani işaretini tahtanın ortasına koyacaktır.

Tic Tac Toe Oyunun karar mekanizmasını ortaya koyan C# kodu EK-C'de verilmiştir.

3.2.1.3 Oyunun çalıştırılması ve arayüz

Oyun C# ile yazılmış bir Windows Form uygulamasıdır. Oyunun gereksinimleri kendi çalıştırılabilir (.EXE) dosyası ve KONTROLLER.DLL dosyasıdır. Oyun çalıştırıldığı zaman Şekil 3.4 ile gösterilen ekran ile karşılaşılır.



Şekil 3.4: Tic Tac Toe Açılış Ekran Görüntüsü

Menü çubuğundaki düğmelerden “Oynat Uğurcuğum” hamle sırasının bilgisayarda olduğunu belirtir, ancak ilk hamleyi insan oyuncu yapmalıdır. Bunun amacı başlangıçtaki arama ağacının bir nebze olsun küçültülmesidir.

İkinci düğme “Yeni Oyun” düğmesi, mevcut oyunu sonlandırır ve oyun tahtasını temizler.

Oyuncu, istediği alana sol faresi ile tıklayarak işaretini koyabilir. Programın yapay zekâsının biraz daha özel durumlarda sınanabilmesi için sağ fare tuşu kullanılarak istenilen yere bilgisayarın işareti konulabilir. Bilgisayarın hamle yapması istendiğinde “Oynat Uğurcuğum” düğmesine basılmalıdır. Böylece program, arama ağacı oluşturup MiniMax algoritmasını işletir.

3.2.2 RPG – bir çatışma oyunu

3.2.2.1 Oyunun ve kurallarının tanımlanması

RPG, “Role Playing Game”in kısaltılmış halidir ve oyun yapı itibariyle tam bir oyun olmaktan çok, yapay zekâlı RPG oyunları için bir yazılım kitaplığı (framework) şeklindedir.

RPG, aslında bir oyun tarzı adıdır. RPG tarzı oyunlarda oyuncular kendilerine bir karakter yaratıp, karakterleri ile hayali bir dünyada geçen bir maceraya atılırlar. RPG tarzı oyunların önemli bir bölüm John Ronald Reuel Tolkien’in kitaplarında anlattığı “Orta Dünya”ya benzeyen diyarlarda geçer. Tolkien tasvirlerine dayanan RPG oyunlarına genellikle FRP (Fantasy Role Playing) adı verilir ve FRP oyunlardaki ırklar Tolkien’in Orta Dünya’sı ile büyük benzerlikler taşır.

3.2.2.1.1 Oyunla ilgili genel bilgiler

Söz konusu oyunun tanımlanması için öncelikle RPG tarz oyunlara özgü bazı terimleri açıklığa kavuşturmak gerekmektedir:

3.2.2.1.1.1 Terimler

AC: Armor Class. Bir yaratığın üzerindekilerin (kendi teni de dahil) onu saldırılara karşı koruma gücü. -10 ile +10 arası bir değer alır ve düşük AC alınacak darbelerin sayısını azaltır.

HP: Hit Point. Bit yaratığın arda kalan toplam can puanları sayısı. Yaratığın aldığı her bir darbe can puanlarından belirli bir miktar götürür ve can puanları sıfır ya da sıfırdan küçük olursa yaratık ölmüştür.

Parti: Aynı amacı güden oyuncular topluluğudur. RPG tarzı oyunlar genellikle bir parti yaratığın başından geçen olaylar olarak anlatılır.

THAC0: To Hit Armor Class Zero. Ham vuruş şansı değeri, yani AC'si sıfır olan bir yaratığa vuruş şansı değeri. 1ile 20 arası bir değerdir ve silah ile kullananın uyumuna göre değişir. Usta bir dövüşçüsünün iyi kullandığı bir silahta THAC0 değeri düşüktür, yani düşük THAC0 vuruş olasılığını arttırır.

Yaratık: Bilgisayar ya da insan, oyunun hikâyesinde yer alan, herhangi bir eylemi gerçekleştirebilen varlıklardır. Yaratıklar, oyunda insan ırkından olabileceği gibi cüce, elf, hatta ejderha ve köpek olabilir.

3.2.2.1.1.2 Kurallar

RPG tarzı oyunlarda karakterlerin birbirleriyle ve çevrelerindeki dünya ile etkileşimlerinin önemli bir bölümü zarlar aracılığıyla yapılır. Genel olarak bir eylemin gerçekleştirilebilmesi için 1-20 arası bir zorluk zarı vardır. Eylemi gerçekleştirmek isteyen oyuncu (yaratık) bir 20 yüzlü zar atar, eğer eyleme ilişkin avantaj ya da dezavantajları varsa atılan zara artı ya da eksi olarak yansıtılır ve eğer sonuç zorluk zarının değerinden büyükse oyuncu eylemi gerçekleştirebilir. Örneğin, AD&D II kurallarında (Advanced Dungeons and Dragons II – bir FRP oyun kural kümesi, oyunumuzda kullanılan kurallar) bir başka yaratığa saldırı esnasında vurmak için atılan 20'lik zar, oyuncunun ham vuruş şansı değerinden büyük olmalıdır. Eğer vuruş başarılı olursa, bu kez de saldırganın vereceği zararı ortaya koymak için saldırganın silahının vuruş gücüne bakılır. Örneğin 2d4 bir sopa için iki kez dörtlük zar atılarak verilecek zarar bulunur ve savunan taraftan bu değer kadar (en az 2, en çok 8) can puanı (HP) eksiltilir. Benzer şekilde vuruş gücü 1d6 olan bir kısa kılıcın vereceği zar 1-6 HP, 2d5 olan bir geniş kılıcın vuruş gücü ise 2-10 HP arasıdır.

3.2.2.1.1.3 Oyunun tanımlanması

RPG, an itibariyle 8x8 bir tahta üzerinde iki düşman partinin karşılaşması ve birbirleriyle savaşmaları şeklinde geçmektedir. Partilerin birbirlerinin düşmanlığından haberdar olmaları, her bir oyuncunun ayrı ayrı kime saldıracağını ortaya koymaları ise yapay zekâ ile gerçeklenmiştir.

Oyun başladığında yaratıklar rasgele yerlere konuşlanırlar ve her “Next Move” düğmesine basıldığında yaratıklar bir sonraki adımlarını hesaplarlar.

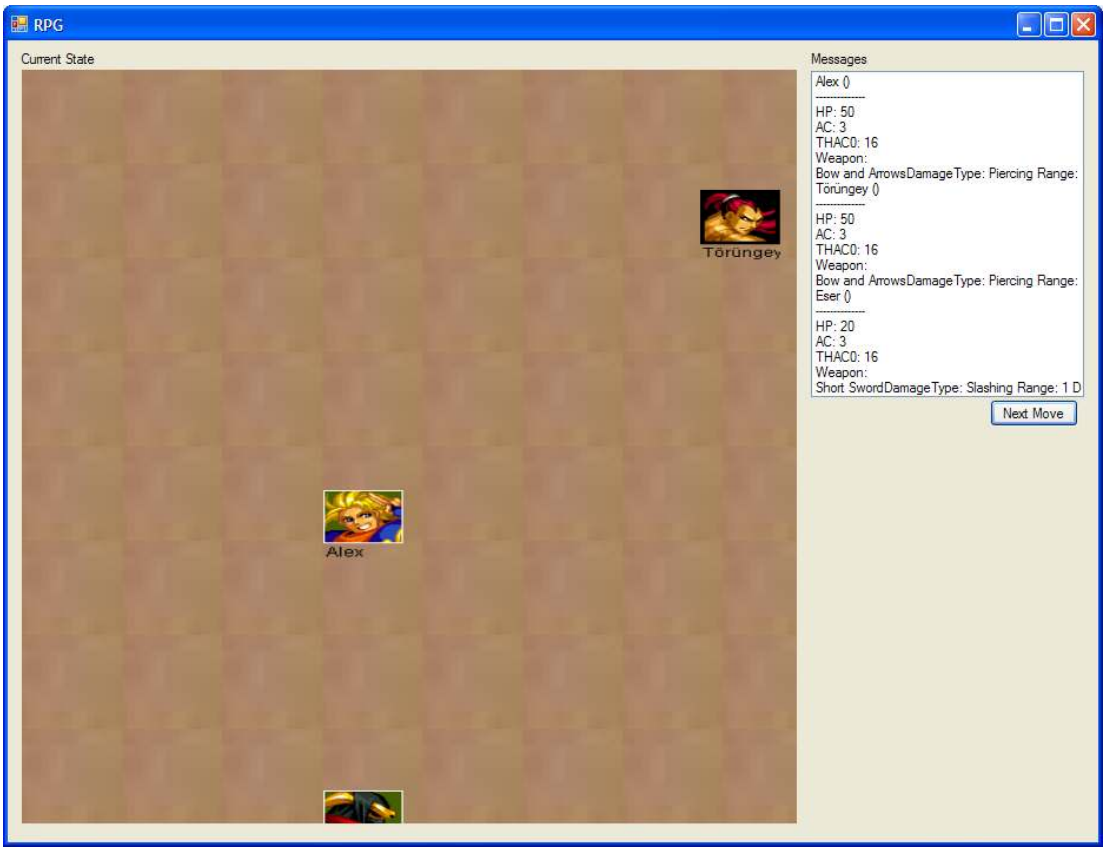
Yapay zekâ motoru her oyuncuyla ayrım gözetmeksizin ayrı ayrı oynayarak en iyi hamleyi bulmaya çalışır ve en iyi olarak değerlendirdiği hamleyi yapar.

Oyun, partilerden herhangi birinin yaratıkları tamamen ölünceye kadar, yani HP değerleri sıfır ya da daha küçük kalana kadar sürer.

3.2.2.2 Oyunun çalıştırılması ve arayüzü

RPG, C#.NET ile yazılmış bir Windows Form uygulamasıdır ve UI.EXE dosyası çalıştırılarak başlatılır. Oyunun yapay zekâsı her bir oyuncunun parametrelerini atamak için bir XML dosyasına gereksinim duyar. Bu XML dosyasının adı AI.CONFIG.XML olmak durumundadır. Örnek bir yapılandırma dosyası EK-D’de bulunabilir. Eğer bir bilgisayar oyuncusu için değer ataması yapılmamışsa, program bu değerlere toplamları 1 edecek şekilde rastgele değerler atar.

Program çalıştırıldığında ekrana gelen görüntü Şekil 3.5’teki gibidir.



Şekil 3.5: RPG Açılış Ekran Görüntüsü

Ekranda görülen büyük alan, 8x8 boyutlarındaki çatışma alanını temsil eder ve üzerindeki her portre bir oyuncuyu belirtir.

Çatışma alanının sağındaki metin kutusu, programın bildirilerinin yazıldığı ileti kutusudur.

İleti kutusunun hemen altındaki “Next Move” düğmesi, programa bir sonraki eli oynamasını söyler.

Kullanıcı “Next Move” düğmesine her bastığında program her bir oyuncu için bir sonraki hamleyi hesaplar ve çatışma alanını yeniden çizer. Partilerden birisine ait bütün oyuncular öldüğünde oyun biter ve “Next Move” düğmesi işe yaramaz.

3.2.2.3 Kullanılan yapay zekâ

RPG,

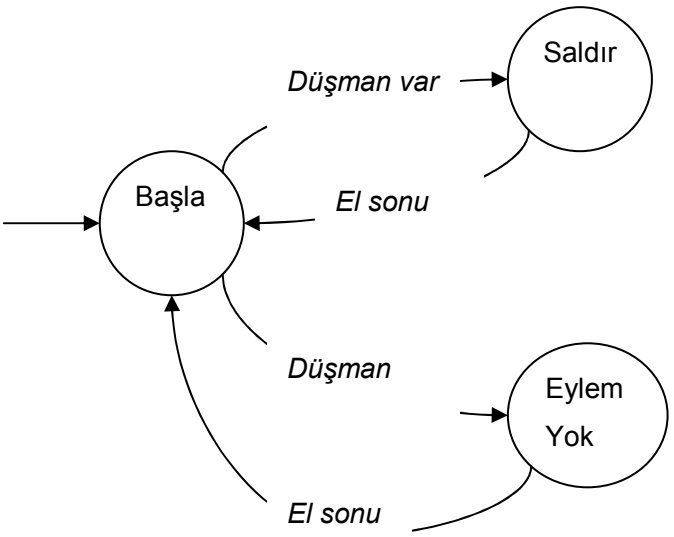
- Eksik bilgili (karşıdaki oyuncuya vurma olasılığı ve verilen zarar gibi rastgele unsurlar içeren)
- Sıra tabanlı
- Sıfırdan farklı toplam bir oyundur.

Ayrıca, RPG'de amaç oyundaki duruma göre değişkendir. Bu yüzden, daha önce bahsedilmiş olan MiniMax yöntemi ile bir çözüme varmak çok zorlaşır. Bu belgede sunulan çözüm, geleneksel çözümlerle ortak noktası bulunan bir başka çözümdür.

3.2.2.3.1 Karar mekanizmaları

3.2.2.3.1.1 RPG'nin karar mekanizması

RPG'de bilgisayarın yapacağı eylemi belirleyen sonlu durum makinesi an itibariyle Şekil 3.6'daki gibidir:



Şekil 3.6: RPG Karar Sonlu Durum Makinesi

3.2.2.3.1.2 Durumlar

Başla: Başlangıç için durum ve yapay zekânın her elin başında bulunduğu durum.

Saldır: Saldırılacak düşmanın seçildiği ve saldırıldığı durum.

Eylem Yok: Herhangi bir eylemin gerçekleştirilmediği durum.

3.2.2.3.1.3 Saldırılacak düşmanın seçilmesi:

Oyunda her oyuncu, kendi sırası geldiğinde bir başka yaratığa saldırır. RPG'de yapay zekâ, saldırılacak düşmanı seçerken her bir düşman için aşağıdaki parametreleri hesaplar:

α: Hedef yaratığın durumu için önceden belirlenmiş bir değer. Varsayılan değerler Denklem 3.2'de verilmiştir. Çatışma ve normal *α* değerlerinin sayısal değerlerinden çok oranları önemlidir. Burada da alınan değerlerde, çatışma halindeki bir rakibin ağırlığının, çatışmayla ilgilenmeyen bir rakibinkinden iki kat fazla olması amaçlanmıştır

$$\alpha_i = \begin{cases} 0.5, & durum_i = çatışma \\ 0.25, & durum_i = normal \end{cases} \quad (3.2)$$

β: Eldeki silahın maksimum vuruş gücüne göre hedef yaratığı öldürmek için geçmesi gerekecek el sayısıdır ve Denklem 3.3'te verilmiştir. Denklem 3.3'teki π sayısı, eldeki silahın maksimum vuruş gücü, *ρ* ise oyuncunun el başına vuruş hakkı sayısını belirtir.

HP_i ve HP_{oyuncu}: Sırasıyla i. rakibin ya da sırası gelen oyuncunun kalan can puanı miktarı.

$$\beta_i = \frac{HP_i}{\pi \cdot \rho} \quad (3.3)$$

γ: Hedef yaratığın yaptığı saldırılarda kaybedilmiş can puanları toplamı.

ε: Hedef yaratığın son saldırısından dolayı kaybedilen can puanı miktarı.

θ: Hedef yaratığın kontrol edilen yaratığa olan uzaklığı ile kontrol edilen yaratığın elindeki silahın menzili. Eğer hedef yaratık menzil içiyse sıfır. Denklem 3.4a'daki δ değeri iki oyuncu arasındaki mesafenin silah menzilinden farkını, *dist* fonksiyonu, iki oyuncu arasındaki mesafeyi, φ ise mevcut silahın menzilini göstermektedir.

$$\delta_i = dist(oyuncu, i) - \varphi \quad (3.4a)$$

$$\theta_i = \max(0, \delta_i) \quad (3.4b)$$

μ: Kontrol edilen yaratığın kalan can puanları miktarının ε'a oranı.

$$\mu_i = \frac{HP_{oyuncu}}{\varepsilon_i} \quad (3.5)$$

Bu parametreleri kullanarak ortaya konan skor her bir düşman yaratık için hesaplanır ve en yüksek skora sahip yaratığa saldırılır. Skor hesaplaması Denklemler (3.6-3.11)'de gösterildiği şekilde yapılır. Denklem 3.11'de s, skoru belirtir.

$$a_i = k_\alpha \cdot \alpha_i \quad (3.6)$$

$$g_i = k_\gamma \cdot \gamma_i \quad (3.7)$$

$$m_i = k_\mu \cdot \mu_i \quad (3.8)$$

$$b_i = k_\beta \cdot \frac{1}{\beta_i} \quad (3.9)$$

$$t_i = k_\theta \cdot \frac{1}{\theta_i} \quad (3.10)$$

$$s_i = a_i + g_i + m_i + b_i + t_i \quad (3.11)$$

3.2.2.4 Yapay zekâ'nın eğitimi

Yapay Zekâ'nın hangi düşmana saldıracağını belirlediği skor fonksiyonun tanımı Denklem (3.6-3.11)'de verilmiştir. Bu fonksiyonun genelleştirilmiş ve açılmış hali Denklem 3.12'de verilmiştir.

$$k_1 \cdot x_{i,1} + k_2 \cdot x_{i,2} + k_3 \cdot x_{i,3} + k_4 \cdot x_{i,4} + k_5 \cdot x_{i,5} = score_i \quad (3.12)$$

Toplamda n adet düşmanı olan bir yaratığın en yüksek skor en üstteki olmak üzere oluşturacağı skor tablosu Denklem 3.13'teki gibidir ($score_i$ yerine kısaltmak için s_i yazılmıştır).

$$\begin{aligned}
&k_1 \cdot x_{1,1} + k_2 \cdot x_{1,2} + k_3 \cdot x_{1,3} + k_4 \cdot x_{1,4} + k_5 \cdot x_{1,5} = s_1 \\
&k_1 \cdot x_{2,1} + k_2 \cdot x_{2,2} + k_3 \cdot x_{2,3} + k_4 \cdot x_{2,4} + k_5 \cdot x_{2,5} = s_2 \\
&\vdots \\
&k_1 \cdot x_{n,1} + k_2 \cdot x_{n,2} + k_3 \cdot x_{n,3} + k_4 \cdot x_{n,4} + k_5 \cdot x_{n,5} = s_n
\end{aligned} \quad (3.13)$$

Bu denklemlerin matris formu ise Denklem (3.14a-3.14b)'de verilmiştir.

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & x_{n,4} & x_{n,5} \end{bmatrix} \cdot \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_n \end{bmatrix} \quad (3.14a)$$

$$X \cdot k = s \quad (3.14b)$$

Yapay Zekâ'nın eğitiminin temelinde saldırılacak yaratığın deneme-yanılmayla bulunmaya çalışılmasıdır. Bunun için algoritma, her bir yaratığın s_i değerinin diğer değerlerden daha yüksek olmasını sağlamak durumundadır. Bunu sağlamak için her bir yaratık için; yaratığın s_i değerini, bilinen k_i'lerce en yüksek kılınan skor olan s_1 ile 0'dan büyük ama 0'a çok yakın bir δ değeriyle toplamına eşitleyip Denklem 3.14a'den yeni bir s vektörü oluşturulur. Bu vektörün s_2'ye göre yapılmış durumu Denklemler (3.15a-3.15b) ile gösterilmiştir.

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & x_{n,4} & x_{n,5} \end{bmatrix} \cdot \begin{bmatrix} k_{2_1} \\ k_{2_2} \\ k_{2_3} \\ k_{2_4} \\ k_{2_5} \end{bmatrix} = \begin{bmatrix} s_1 \\ s_1 + \delta \\ s_3 \\ s_4 \\ \vdots \\ s_n \end{bmatrix} \tag{3.15a}$$

$$X \cdot k_2 = s_2 \tag{3.15b}$$

Yeni denklemde s_2 vektörünün 2. satırdaki değerin en yüksek değer olacağı aşikâr olduğu için skor fonksiyonunun katsayıları Denklem 3.15b'de verilen k_2 vektörünü skor listesinde zirveye taşıyacaktır. 2. satırdaki değerlere sahip olan yaratığı ilk saldırılacak yaratık kılacak skor fonksiyonu ağırlıklarını oluşturan sayılardan oluşan k_2 vektörünü elde etmek için Denklemler (3.16a-3.16b) kullanılır.

$$X^{-1} \cdot X \cdot k_2 = X^{-1} \cdot s_2 \tag{3.16a}$$

$$k_2 = X^{-1} \cdot s_2 \tag{3.16b}$$

Denklem 3.16b, X matrisinin kare matris olmadığı durumlarda kare olmayan matrisler birer tekil matris olduğu için kullanılamaz. Bu yüzden Denklem 3.16b genelleştirilmelidir. Kare olmayan matrisler içeren denklemlerde matris sözde ters (Moore-Penrose tersi) ile bulunur. K_2 vektörünü yalnız bırakmak için denklemin her iki tarafı da soldan X matrisinin sözde tersi ile çarpılır. Sözde ters ile X matrisinin çarpılmış durumu Denklemler (3.17a-3.17b)'de gösterilmiştir.

$$X^+ \cdot X \cdot k_2 = X^+ \cdot s_2 \quad (3.17a)$$

$$k_2 = X^+ \cdot s_2 \quad (3.17b)$$

Moore-Penrose tersi; -kare olsun olmasın- her matris için mevcuttur ve matrisin – eğer varsa- gerçek, soldan ya da sağdan tersini verir. Bir matrisin Moore-Penrose tersi, o matrisin tekil değer ayrışımından yola çıkılarak bulunur [17]. m x n boyutlarındaki bir A matrisinin tekil değer ayrışımı Denklem 3.18'de verilmiştir.

$$A = U \cdot D \cdot V^* \quad (3.18)$$

Denklemde; U, m x n boyutlarında bir üniter matris, V* n x n boyutlarında üniter bir matris olan V'nin tümleyen (konjuge) transpozesi, D ise m x n boyutlarında, köşegen üzerinde sıfırdan büyük sayılar içeren, köşegen dışındaki sayıları sıfır olan bir matristir. V ile U matrisleri birbirlerine göre ortonormaldir.

A matrisinin Moore-Penrose tersi, Denklem 3.19'da verilmiştir.

$$A^+ = V \cdot D^+ \cdot U^* \quad (3.19)$$

Denklem 3.17b kullanılarak K_2 vektörü çözüldükten sonra oyun yapay zekâ tarafından K_2 vektörünün elemanları yeni katsayılar olarak alınarak baştan sona kadar oynanır. Oyunun başlangıç koşulları yazılımın yapılandırma dosyasında belirlenmiştir. Oyun sona erdiğinde yapay zekâ'nın K_2 vektörüyle elde ettiği başarıyı belirlemek için bir değerlendirme fonksiyonu kullanılır. Örnek bir değerlendirme fonksiyonu denklem 3.20'de verilmiştir. HP_{parti1}, eğitilen yapay zekâ oyuncusunun partisinde bulunan yaratıkların (dostların) can puanlarını belirtirken HP_{parti2}, karşı partide bulunan yaratıkların (düşmanların) can puanlarını belirtir.

$$E_2 = \sum HP_{parti\,1} - \sum HP_{parti\,2} \quad (3.20)$$

Eğitimin bir sonraki adımında, eğitim başlangıcındaki skor listesine göre 3. sırada olan yaratığın skoru en yükseğe taşınmaya çalışılır. Bunun için de Denklem 3.15a'ya benzer bir denklem olan Denklem 3.21a oluşturulur.

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & x_{n,4} & x_{n,5} \end{bmatrix} \cdot \begin{bmatrix} k_{3_1} \\ k_{3_2} \\ k_{3_3} \\ k_{3_4} \\ k_{3_5} \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_1 + \delta \\ s_4 \\ \vdots \\ s_n \end{bmatrix} \tag{3.21a}$$

$$X \cdot k_3 = s_3 \tag{3.21b}$$

k_3 vektörünü çözümlemek için yine Moore-Penrose tersi yöntemi kullanılır ve denklem 3.22b elde edilir.

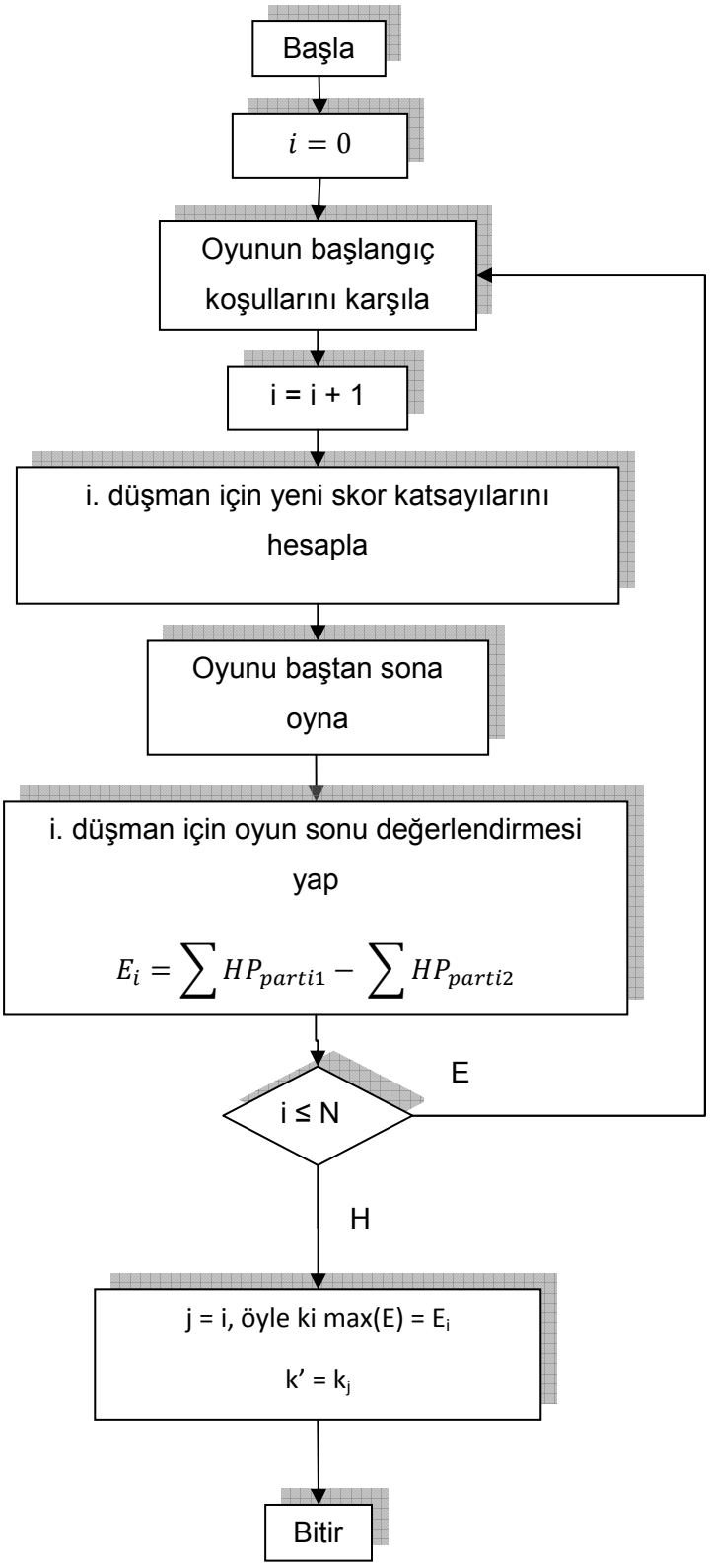
$$X^+ \cdot X \cdot k_3 = X^+ \cdot s_3 \tag{3.22a}$$

$$k_3 = X^+ \cdot s_3 \tag{3.22b}$$

Yapay zekâ, skor fonksiyonu katsayılar olarak k_3 vektörünün elemanlarını kullanarak oyunu baştan sona oynar ve oyun sonunda Denklem 3.23 ile E_3 değerini elde eder.

$$E_3 = \sum HP_{parti1} - \sum HP_{parti2} \tag{3.23}$$

Değerlendirme fonksiyonu değerlerinin en büyüğü olan değeri sağlayan k vektörü eğitimin en iyi değeri olarak kabul edilir. Eğitimin algoritma akış çizelgesi Şekil 3.7'de verilmiştir.



Şekil 3.7: RPG Yapay Zekâ Eğitim Algoritması

3.2.2.4.1 N < 5 durumu

Skor tablosundaki düşman sayısının skor fonksiyonu katsayıları sayısından, yani 5'ten küçük olduğu durumlarda 5 elemanlı bir vektörü elde etmek için 5'ten az sayıda denklem kullanmanın yetersiz olacağından ötürü, bu durumlarda k vektörünü duyarlılık analizi ile kırpmak gerekir.

Duyarlılık analizinde, k vektöründeki her bir elemanın s vektörü değerlerini ne kadar değiştirdiğini bulmak için k vektöründeki her bir eleman belirli bir küçük sayı oranında arttırılır. Değişimin oransal olması önemlidir; çünkü duyarlılık analizindeki sonuç s vektörünün eski durumuna göre oranıdır. Duyarlılığı belirli bir oranın altında kalan k vektörü değerleri, s vektörü değerlerini çok etkilemedikleri için göz ardı edilir, vektörden o eğitim için koparılır. Yeni denklem sistemi ve matris ile vektörlerin boyutları denklem 3.24'te verilmiştir.

$$X'_{(m\times m)} \cdot k'_{(m\times 1)} = s'_{(m\times 1)} \tag{3.24}$$

Denklem 3.24'te de görüldüğü gibi, s vektörünün boyutlarında bir değişim olmamaktadır.

k vektöründeki bir eleman olan k_i ($i \leq 5$)'nin s vektörünü etkileme miktarını bulmak için k_i sayısı kendisinin 0,01 katıyla çarpılır ve toplam skora olan katkısı bulunur. Denklemler (3.25a-3.25c), i = 1 için duyarlılık analizini gösterir.

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & x_{n,4} & x_{n,5} \end{bmatrix} \cdot \begin{bmatrix} k_1 + 0.01 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix} = \begin{bmatrix} s_1^1 \\ s_2^1 \\ s_3^1 \\ s_4^1 \\ \vdots \\ s_n^1 \end{bmatrix} \tag{3.25a}$$

$$X \cdot k_1' = s_1' \tag{3.25b}$$

$$C_1 = \left| 1 - \frac{\sum s_i}{\sum s_i^1} \right| \tag{3.25c}$$

Her 5 değer için de C_i değerleri bulunduktan sonra en düşük (5 – n) adet katsayı değeri formülden çıkartılmak üzere denklem 3.26 elde edilir.

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & x_{n,n} \end{bmatrix} \cdot \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_n \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_n \end{bmatrix} \tag{3.26}$$

Bu aşamadan sonra algoritma denklem 3.15b gibi sürdürülür. Duyarlılık analizi skor fonksiyonunun değerlerini görmezden geldiği için eğitimin kalitesinin düşme olasılığı vardır. Bu yüzden eğitimlerin başarısı “5 ya da daha fazla düşman” ve “5’ten az düşman” olarak iki ayrı yoldan izlenmelidir.

4 BULGULAR VE TARTIŞMA

4.1 Kullanılan Yöntemlerin Başarısı

4.1.1 Tic tac toe

MiniMax algoritmasının etkinliği Tic Tac Toe oyununda açıkça görülebilir. Bilgisayar, insana karşı oyun kaybetmemektedir. Buna karşın 3x3'lük bir oyun tahtasının bile ne kadar çok işlem gücü gerektirdiği oyundaki ilk hamlenin bekleme süresinden de anlaşılmaktadır. Örnek bir oyun Şekil 4.1 – Şekil 4.9 arası verilmiştir.

Oyunun açılışını insan oyuncu yapmıştır ve işaretini 5. kareye koymayı tercih etmiştir.

Şekil 4.1: Tic Tac Toe Hamle 1(İnsan)

Yapılan hamleye karşılık olarak yapay zekâ, oyuncunun 1. çaprazdan oyunu kazanmasını engellemek ve 1. yatay sıra ile 1. düşey sıradan oyunu kazanabilmek için işaretini 1. kareye koymuştur. Bilgisayarın yaptığı ilk hamle olan hamle 2 Şekil 4.2'de gösterilmiştir.

Şekil 4.2: Tic Tac Toe Hamle 2(Bilgisayar)

Bilgisayarın yaptığı hamleye karşılık olarak insan oyuncu, ikinci çaprazdan sayıya gitmek üzere 3. kareye işaretini koymuştur. İnsan oyuncunun 2. hamlesi üzerine oyunun durumu Şekil 4.3'te verilmiştir.

Şekil 4.3: Tic Tac Toe Hamle 3 (İnsan)

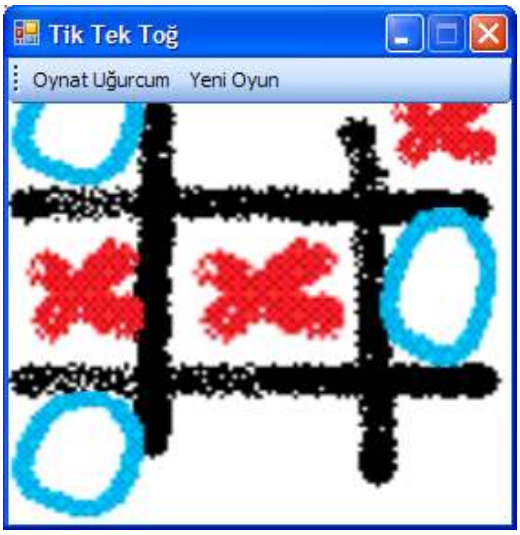
Yapay zekâ oyuncusu rakibinin sayı yapmasını engellemek ve 1. düşey sıradan sayıya gitmek için işaretini 7. kareye bırakır ve oyun Şekil 4.4'teki duruma gelir.

Şekil 4.4: Tic Tac Toe Hamle 4 (Bilgisayar)

İnsan oyuncu, bilgisayarın sayısını engellemek ve 2. yatay sıradan kendi sayısına ulaşabilmek için işaretini 4. kareye koyar. Oyundaki 5. hamle olan bu hamleyle oyunun durumu Şekil 4.5'teki gibi olur.

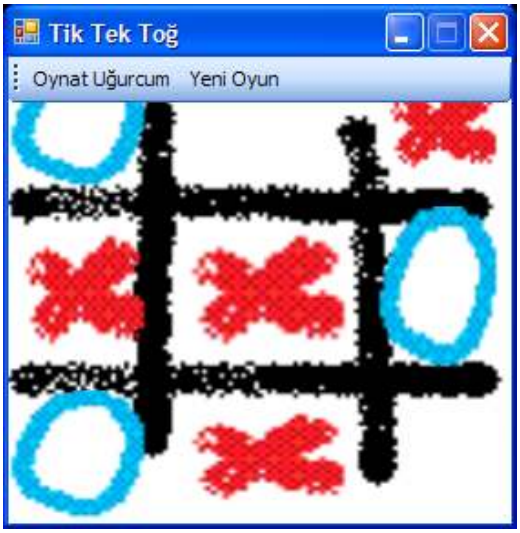
Şekil 4.5: Tic Tac Toe Hamle 5 (İnsan)

İnsan rakibinin sayısına engel olmak isteyen program, işaretini 6. kareye koyarak oyunu Şekil 4.6'daki duruma getirir.



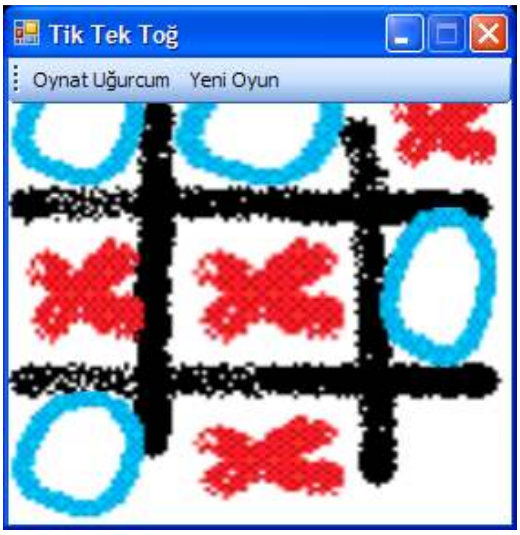
Şekil 4.6: Tic Tac Toe Hamle 6 (Bilgisayar)

Son sayı şansını da kullanmak isteyen insan oyuncu 4. işaretini 8. kareye koyar. Son durum şekil 4.7'de verilmiştir.



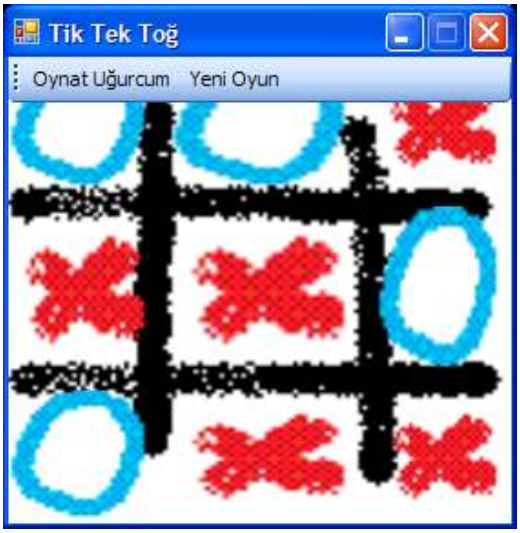
Şekil 4.7: Tic Tac Toe Hamle 7 (İnsan)

İnsan rakibinin son sayısını da engelleyecek olan program, işaretini 2. kareye koyarak oyundaki beraberliğin kesinleşmesini sağlar. Oyundaki 8. hamle olan bu hamle sonunda oyun tahtası Şekil 4.8'deki gibidir.



Şekil 4.8: Tic Tac Toe Hamle 8 (Bilgisayar)

İnsan oyuncunun 9. kareden başka kullanabileceği kare kalmadığından işaretini 9. kareye koyar ve böylece oyun berabere sonuçlanmış olur.



Şekil 4.9: Tic Tac Toe Hamle 9 (İnsan)

Şekiller 4.4, 4.6 ve 4.8'de görüldüğü gibi program, insanın her sayı teşebbüsünü engellemekte ve oyuncuya yenilmemektedir. Bunun nedeni, programın MiniMax yöntemini kullanarak oyunu baştan sona defalarca oynaması ve kendisi için en iyi sonuca yönelik olan hamleyi yapmasıdır. MiniMax algoritması en yüksek kaybı minimize ettiğinden insan oyuncu hiç hata yapmadan oynar ve oyunu 8. hamleye kadar devam ettirebilse bile yapay zekâ; insana yenilmeyecek, berabere kalarak kazancı olmamasına rağmen kaybını da sıfırlayacaktır. Eğer insan oyuncu yanlış bir hamle yaparsa yapay zekâ en iyi oyununu oynayarak bu sefer kazancını maksimize etmeye, yani en kısa sürede oyunu kazanmaya çalışacaktır.

Tic tac toe'daki değerlendirme fonksiyonu kazanılan oyunlarda 0'dan büyük, kaybedilen oyunlarda 0'dan küçük ve berabere kalınan oyunlarda 0 ürettiği için yapay zekâ; kazanmayı beraberliğe ve yenilgiye, beraberliği ise yenilgiye tercih etmekte ve buna göre hamle yapmaktadır.

4.1.2 RPG

RPG'nin başarımını ölçmek için farklı yapılandırmaları olan yapay zekâ oyuncuları birbirleriyle savaştırılır. Her yapılandırmada, Denklemler (3.6-3.10) arası verilen sabit katsayılar değiştirilir. Her test, başlangıç koşulları farklı olan eş güçteki oyuncuları sınar.

4.1.2.1 Başlangıç koşulları

Başlangıç koşullarında, oyunda iki adet düşman parti bulunmaktadır ve partilerde bulunan yaratıklara dair ayrıntılı bilgiler şöyledir:

Parti 1

1. AC = 3, HP = 50, THAC0 = 16, Silah = Ok/Yay (1d10, menzil = 10)
2. AC = 3, HP = 20, THAC0 = 16 Silah = Kısa kılıç (2d5, menzil = 1)

Parti 1'deki yapay zekâ oyuncularının değerlendirme fonksiyonu sabit katsayıları 0.2 olarak belirlenmiştir.

Parti 2

1. AC = 3, HP = 50, THAC0 = 16, Silah = Ok/Yay (1d10, menzil = 10)
2. AC = 3, HP = 20, THAC0 = 16 Silah = Kısa kılıç (2d5, menzil = 1)

Parti 2'deki yapay zekâ oyuncularının değerlendirme fonksiyonu sabit katsayıları her oyun başında rastgele atanmaktadır.

4.1.2.2 Test sonuçları

Tablolar (4.1-4.6)'da kullanılan, (✓) işareti, o oyunun galibinin sütununa konmuştur.

2'ye 2 oynanan ve her bir oyuncunun yeri önceden belirlenmiş olan 10 ardışık oyunun test sonuçları Tablo 4.1'de gösterilmiştir. Sonuçlara göre katsayıları rastgele seçilen Parti 2, %80 başarı sağlamıştır.

Tablo 4.1: 2'ye 2 Sabit Konumlu Test Sonuçları

Oyun	Parti 1 (Sabit Katsayılı)	Parti 2 (Rastgele Katsayılı)
1		✓
2		✓
3		✓
4		✓
5	✓	
6		✓
7		✓
8		✓
9	✓	
10		✓

2'ye 2 oynanan ve her bir oyuncunun başlangıç konumu rastgele olarak belirlenmiş 10 ardışık oyunun test sonuçları Tablo 4.2'de verilmiştir. Sonuçlara göre Parti 2, yine %80 başarı sergilemiştir.

Tablo 4.2: 2'ye 2 Rastgele Konumlu Test Sonuçları

Oyun	Parti 1 (Sabit Katsayılı)	Parti 2 (Rastgele Katsayılı)
1		✓
2		✓
3		✓
4	✓	
5		✓
6	✓	
7		✓
8		✓
9		✓
10		✓

Parti 1'den 1, Parti 2'den 2 oyuncunun oynadığı ve her bir oyuncunun başlangıç konumunun sabit olduğu 5 ardışık oyunun test sonuçları Tablo 4.3'te verilmiştir. 1'e 2 çatışmada sayıca az olan tarafın kazanma olasılığı düşük olduğundan, tabloya başarı ölçütü olarak diğer partideki oyuncuların kalan can puanları eklenmiştir.

Tablo 4.3: 1'e 2 Sabit Konumlu Test Sonuçları

Oyun	Parti 1 (Sabit Katsayılı, 1 Oyuncu)	Parti 2 (Rastgele Katsayılı, 2 Oyuncu)	Parti 2 Oyuncu 1 Kalan Can	Parti 2 Oyuncu 2 Kalan Can
1		✓	15	12
2		✓	24	17
3		✓	31	16
4		✓	10	20
5		✓	11	10

Parti 1'den 1, Parti 2'den 2 oyuncunun oynadığı ve her bir oyuncunun başlangıç konumunun rastgele olduğu 5 ardışık oyunun test sonuçları Tablo 4.4'te verilmiştir. 1'e 2 çatışmada sayıca az olan tarafın kazanma olasılığı düşük olduğundan, tabloya başarı ölçütü olarak diğer partideki oyuncuların kalan can puanları eklenmiştir.

Tablo 4.4: 1'e 2 Rastgele Konumlu Test Sonuçları

Oyun	Parti 1 (Sabit Katsayılı, 1 Oyuncu)	Parti 2 (Rastgele Katsayılı, 2 Oyuncu)	Parti 2 Oyuncu 1 Kalan Can	Parti 2 Oyuncu 2 Kalan Can
1		✓	26	16
2		✓	36	20
3		✓	38	11
4		✓	29	20
5		✓	26	10

Parti 1'den 2, Parti 2'den 1 oyuncunun oynadığı ve her bir oyuncunun başlangıç konumunun önceden belirlenmiş olduğu 5 ardışık oyunun test sonuçları Tablo 4.5'te verilmiştir. 1'e 2 çatışmada sayıca az olan tarafın kazanma olasılığı düşük olduğundan, tabloya başarı ölçütü olarak diğer partideki oyuncuların kalan can puanları eklenmiştir.

Tablo 4.5: 2'ye 1 Sabit Konumlu Test Sonuçları

Oyun	Parti 1 (Sabit Katsayılı, 2 Oyuncu)	Parti 2 (Rastgele Katsayılı, 1 Oyuncu)	Parti 1 Oyuncu 1 Kalan Can	Parti 1 Oyuncu 2 Kalan Can
1	✓		28	1
2	✓		27	14
3	✓		3	5
4	✓		19	15
5	✓		14	15

Parti 1'den 2, Parti 2'den 1 oyuncunun oynadığı ve her bir oyuncunun başlangıç konumunun rastgele olduğu 5 ardışık oyunun test sonuçları Tablo 4.6'da verilmiştir. 1'e 2 çatışmada sayıca az olan tarafın kazanma olasılığı düşük olduğundan, tabloya başarı ölçütü olarak diğer partideki oyuncuların kalan can puanları eklenmiştir. Parti 2'deki tek oyuncu, oyunlardan birini kazanmayı başarmıştır.

Tablo 4.6: 2'ye 1 Rastgele Konumlu Test Sonuçları

Oyun	Parti 1 (Sabit Katsayılı, 2 Oyuncu)	Parti 2 (Rastgele Katsayılı, 1 Oyuncu)	Parti 1 Oyuncu 1 Kalan Can	Parti 1 Oyuncu 2 Kalan Can
1	✓		14	16
2		✓	0	0
3	✓		30	12
4	✓		7	20
5	✓		24	14

5 SONUÇLAR VE YORUM

Yapay zekâ, günümüzdeki bilgisayarların daha çok ve daha çeşitli sorunlarla başa çıkabilmesini sağlayan optimizasyonların bir araya getirilmesidir. Karmaşık dünya sorunları matematiksel olarak ele alınıp, bir bilgisayar üzerine programlanabilir duruma getirilerek ortaya konulan yapay zekâ günümüzde bilgisayar bilimlerinin en çok gelişime açık dalları arasındadır ve gittikçe yaygınlığını arttırmaktadır.

Bilgisayar oyunlarında yapay zekâ ise temel olarak insanları eğlendirebilecek düzeyde onlarla mücadele edebilen programlar geliştirmek olarak tanımlanabilir; çünkü hiç kazanamadıkları oyunları oynamak insanlara bir süre sonra sıkıcı gelebilir. Bu yüzden oyun üreticileri genellikle oyunları için yazdıkları yapay zekâ programlarını belirli değişkenlere bağlayarak kullanıcının isteği üzerine yapay zekâyı köreltecek ya da zekileştirecek şekilde bu değişkenleri değiştirmeyi seçerler.

Elbette bir bilgisayar oyunu yalnızca yapay zekâ programlamasından ibaret değildir, ancak bu çalışmada oyun programlamasının ortaya konmak istenen kısmı yapay zekâdır. Bu yüzden elinizde bulunan bu çalışmada yapay zekâda kullanılabilecek yöntemlerin bir kısmı sıralanmış ve iki adet programlama örneği ile örneklenmeye çalışılmıştır.

İlk örnekte verilmiş olan Tic Tac Toe oyunu, insan için oynaması basit bir oyunun bile bilgisayarca ne kadar zor olduğunu ve tam bilgili sıfır toplamlı oyunlarda doğru yapay zekâ programlamasıyla bilgisayarın nasıl yenilemez olduğunu göstermeyi amaçlamaktadır.

Tic Tac Toe, MiniMax algoritmasını kullanan basit bir uygulamadır ve ardışık 10 oyun süresince iyi bir insan Tic Tac Toe oyuncusuna yenilmemiştir. Bunun nedeni, 4. bölümde de verildiği üzere, MiniMax algoritmasının oyunun sonuna kadar olası bütün hamleleri hesaplayarak aralarından bilgisayarın kazancını en yükseğe çıkaran ya da kaybını en aza indirgeyen hamleyi ortaya koymasıdır. Ardışık 10 oyunun sonuçları ve programın her bir hamleyi hesaplamak için harcadığı süreler Tablo 5.1'de verilmiştir.

Tablo 5.1: Tic Tac Toe Test Sonuçları

Oyun	1.Hamle Süresi (ms)	2. Hamle Süresi (ms)	3. Hamle Süresi (ms)	4. Hamle Süresi (ms)	Kazanan
1	918	18	0	0	Berabere
2	952	26	1	0	Berabere
3	8903	14	0	0	Berabere
4	1069	23	0	0	Berabere
5	1049	22	1	0	Berabere
6	1041	27	0	-	Bilgisayar
7	1072	20	3	0	Berabere
8	918	14	-	-	Bilgisayar
9	932	15	0	0	Berabere
10	1014	13	0	0	Berabere

İkinci örnek olan RPG oyununda, sıfır toplam olmaması ve rastgelelik içerdiğinden bir tam bilgili oyun olmaması nedeniyle, bir değerlendirme fonksiyonu kullanılarak saldırılacak düşmanın seçilmesi sorunu aşılmaya çalışılmış ve önemli ölçüde başarı sağlanmıştır. Test sonuçlarına göre Denklemler (3.6-3.10) arası verilmiş olan sabit katsayılar programın hedefini seçmesinde önemli rol oynamaktadır ve en doğru kararı verebilmesi için yapay zekânın bu katsayıları optimize etmesi şarttır.

Bu katsayıların en doğru şekilde güncellenmesi için yapay zekâ bölüm 3.2.2.4'te anlatılan eğitim yöntemi ile eğitilerek insan oyuncuların karşısına çıkmaya hazır duruma getirilebilir. Bu çalışmada sunulan eğitim yöntemi, matris işlemlerine dayanmaktadır ve belirlenmiş yapılandırmaya sahip bir ortamda en iyi sonucu verecek olan katsayıları bulabilir.

Bu tezin içindeki yazılımların tamamı C# programlama diliyle yazılmıştır. Ancak, gerek ücretsiz açık kaynaklı kütüphanelerin sayısı ve kalitesi, gerek uygulama geliştirme ortamlarının darlığı nedeniyle; böylesi bir çalışmada mecbur kalınmadıkça C#'ın kullanılmaması önerilir. Alternatif programlama dilleri olarak JAVA ve LISP incelenebilir.

5.1 İleri Çalışmalar

MiniMax algoritması, yapısı itibariyle işletilmesi zor bir algoritmadır ve Tic Tac Toe oyununda yalın MiniMax'in yavaşlığı özellikle ilk hamlelerde hissedilmektedir.

Tic Tac Toe'nun hızlandırılması için aynı oyunları temsil eden ağaç dalları birden çok kez üretilmek yerine, bu oyunları çıkartan farklı oyun yollarının hepsi bu eş oyunlara bağlanılabilir. Bu yönteme "eş oyunları bulma" yöntemi adı verilir.

İkinci bir yöntem ise alfa-beta kesintilerini kullanmak olabilir. Alfa-beta kesintileri, uygun dizilmiş oyun ağaçlarında çok önemli zaman ve işlem kazancı sağlayabilir. leAncak alfa-beta kesintilerinin verimli çalışabilmesi için öncelikle oyun ağacının uygun şekilde sıralanması gerekmektedir. Bu sıralama için de başka algoritmaların koşturulması gerekmektedir.

Üçüncü bir yöntem, programın gideceği hamle derinliğini azaltmaktır. Böylece MiniMax algoritması oyunun bittiği hamlelere kadar değil, mevcut durumdan belirli sayıda ileriye kadar işletilir ve değerlendirme fonksiyonu o noktada işletilerek yine en yüksek MAX değeri ya da en düşük MİN değeri üzerinden en kârlı hamle bulunmaya çalışılabilir. Ancak unutulmamalıdır ki bu yöntem önemli ölçüde zamandan tasarruf sağlasa da, yapay zekâyı köreltecek bir yöntemdir. Bir hamle derinliği yapay zekâyı aptal kılacak değişikliklere neden olabilir.

RPG'nin eğitimi belirli senaryolar üzerine yapıldığından dolayı, eğitimin ortaya koyacağı katsayı değerlerinin en uygun olduğu durumların sayısı sınırlıdır. Örneğin 2'ye 2 bir eğitim setiyle eğitilmiş bir yapay zekâ, 3'e 3'lük bir oyunda yanlış kararlar verebilir. Bunun önüne geçmek için, RPG yapay zekâsının eğitim adımları, her bir hamleden önce uygulanabilir. Böylece program her hamlesinden önce kendisini bulunduğu duruma göre eğitir ve fonksiyon katsayılarını her hamlesinden önce yeniden hesaplayabilir.

Böylesi bir eğitim kullanıldığında RPG'nin yöntemi, her el başında bütün oyunu birkaç kez (düşman sayısı kez) oynayacağından biraz da olsa MiniMax'i andıracaktır. Ancak, böylesi bir "anında eğitim"in bilgisayara matris hesaplarında ciddi bir ek yük getireceği de göz ardı edilemez.

KAYNAKLAR

[1] McCarthy, J. *Basic Questions. What is Artificial Intelligence,* [Online], http://www-formal.stanford.edu/jmc/whatisai/node1.html, 2007

[2] Jr., Jackson P.C. *Introduction to Artificial Intelligence*, ***Dover***, 2-4, (1985)

[3] Applications of Artificial Intelligence, *Wikipedia* [Online], http://en.wikipedia.org/wiki/Applications_of_artificial_intelligence, 2008

[4] Brownlee, J., *Finite State Machines*, Ai Depot. [Online] http://ai-depot.com/FiniteStateMachines/FSM.html

[5] Savage, L. J. "The Foundations of Statistics", New York, ***Wile***, (1954)

[6] R., Olivas. *A Primer for Decision-making Professionals* [Online] http://www.projectsphinx.com/decision_trees/index.html, 2007

[7] Pinto, P., *Minimax Explained*, Ai Depot, [Online], http://ai-depot.com/articles/minimax-explained/, 2002

[8] *Artificial Neural Networks - A Neural Network Tutorial*. Artificial Neural Networks - A Neural Network Tutorial [Online], http://www.learnartificialneuralnetworks.com/, 2008

[9] Nose Picking using Neural Networks. Karthig's Log. [Online], http://karthik3685.wordpress.com/2007/11/03/nose-picking-using-neural-networks/, 2007

[10] Becerikli, Y., Yapay Sinir Ağlarına Giriş Ders Notları, (2005)

[11] R., Rojas. "Neural Networks", ***Springer-Verlag***, 151-174, 1996.

[12] Z., Ghahramani. "Unsupervised Learning", 2004.

[13] Karakuzu C., Gürbüzer G., "Single Target Tracking Using Adaptive Neuro-Fuzzy Inference Systems", 2006.

[14] da Silva Borges, P.S. *An Application of the Fuzzy Iterated Prisoner's Dilemma*. A model of strategy games based on the paradigm of the Iterated Prisoner's Dilemma employing Fuzzy Sets. [Online], http://www.eps.ufsc.br/teses96/borges/cap6/cap6.htm, 1996.

[15] Rowland T., Weisstein E. W. *Genetic Algorithm*. From MathWorld--A Wolfram Web Resource. [Online] http://mathworld.wolfram.com/GeneticAlgorithm.html .

[16] Saloky T., Šeminský J."Artificial Intelligence and Machine Learning", ***SAMI 2005***, 21 Ocak, 2005

[17] Petersen K.B., Pedersen M.S. "The Matrix Cookbook", 17-18, (2005)

EKLER

EK-A MiniMax sözde kodu

```
MinMax (GamePosition game) { return MaxMove (game); }


MaxMove (GamePosition game) {
if (GameEnded(game)) {
  return EvalGameState(game);
}
else {
  best_move < - {};
  moves <- GenerateMoves(game);
  ForEach moves {
    move <- MinMove(ApplyMove(game));
    if (Value(move) > Value(best_move)) {
      best_move < - move;
    }
  }
  return best_move;
}

}


MinMove (GamePosition game) {
best_move <- {};
moves <- GenerateMoves(game);
ForEach moves {
  move <- MaxMove(ApplyMove(game));
  if (Value(move) > Value(best_move)) {
```

```
    best_move < - move;
  }
}
return best_move;

}
```

EK-B Genetik Algoritma

1. [Başlat] n kromozomlu (çözüm adaylı) rastgele bir nüfus oluştur.
2. [Uygunluk] Her x kromozomu için $f(x)$ uygunluğunu değerlendir.
3. [Yeni nüfus] Yeni nüfus oluşana kadar,
    a. [Seçilim] Uygunluklarına göre iki ebeveyn kromozom seç.
    b. [Çaprazlama] Çaprazlama olasılığı kullanarak, her iki ebeveynden bir yeni çocuk türet. Eğer çaprazlama yapılmazsa, çocuklar ebeveynlerinin tam birer kopyası olur.
    c. [Mutasyon]Mutasyon olasılığı kullanarak, yeni çocukların her bir değişkenini değiştir.
    d. [Kabullenme] Yeni çocukları yeni nüfusun içine koy.
4. [Güncelleme] Türetilmiş nüfusu kullan.
5. [Sınama] Eğer sonlandırma koşulu karşılandıysa dur ve eldeki nüfusun arasından en iyi çözümü dön.
6. [Döngü] 2. adıma git.

EK-C Tic Tac Toe Oyunu Yapay Zekâ Kodu

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using TicTacToe.Kontroller;

namespace TicTacToe
{
    public class OyunBittiEventArgs : EventArgs
    {
        public Oyuncular Kazanan;

        public OyunBittiEventArgs(Oyuncular kazanan)
        {
            Kazanan = kazanan;
        }
    }

    public delegate void OyunBittiDelegate(object sender, OyunBittiEventArgs e);

    class AramaAgaci
    {
```

```
private List<AramaAgaci> _Cocuklar;
private AramaAgaci _Baba = null;
private Oyuncular[,] _Oyun;
private int _Deger;
private int _Nesil;
private Oyuncular _Oynayan;

/// <summary>
/// Oyunu oynayan oyuncuyu belirtir.
/// </summary>
public Oyuncular Oynayan
{
    get { return _Oynayan; }
    set { _Oynayan = value; }
}



/// <summary>
/// Yeni bir arama ağacı nesnesi yaratır
/// </summary>
/// <param name="oyun">Oyun tahtasının anlık durumu</param>
public AramaAgaci(Oyuncular[,] oyun)
{
    _Cocuklar = new List<AramaAgaci>(0);
```

```
    _Oyun = oyun;
    _Nesil = 0;
}

/// <summary>
/// Arama ağacının çocuklarına bir yenisini ekler
/// </summary>
/// <param name="oyun">Eklenecek çocuğun oyun durumu</param>
/// <returns>Eklenen çocuk</returns>
public AramaAgaci CocukEkle(Oyuncular[,] oyun)
{
    AramaAgaci a = new AramaAgaci(oyun);
    _Cocuklar.Add(a);
    a.Baba = this;
    a._Nesil = this._Nesil + 1;
    return a;
}

/// <summary>
/// Arama ağacı düğümünün babası (üst düğümü)
/// </summary>
public AramaAgaci Baba
{
    get { return _Baba; }
```

```
    set { _Baba = value; }
}
/// <summary>
/// Arama ağacının o anki oyununun değeri
/// </summary>
public int OyunDegeri
{
    get { return _Deger; }
    set { _Deger = value; }
}

/// <summary>
/// Arama ağacının o anki oyun durumu
/// </summary>
public Oyuncular[,] Oyun
{
    get { return _Oyun; }
}

public List<AramaAgaci> Cocuklar
{
    get { return _Cocuklar; }
}
```

```
        public bool Babadir
        {
            get { return _Baba == null; }
        }

        public bool Torundur
        {
            get { return _Cocuklar.Count == 0; }
        }

        public int Nesil
        {
            get { return this._Nesil; }
        }
    }

    class Yz
    {
        /// <summary>
        /// YZ motorunun bilgilerinin trace ekranına verilip verilmeyeceğini belirler. İlk değeri true'dur.
        /// </summary>
        public bool TraceMessages = true;
        public event OyunBittiDelegate OnOyunBitti;
```

```
private readonly int _TahtaBoyu;
private readonly int _MaxNesil;
private readonly int _DegerCarpan;
private readonly int _BeraberlikDegeri = 0;

/// <summary>
/// Yeni bir Yapay Zeka Motoru nesnesi yaratır
/// </summary>
/// <param name="oyun"></param>
public Yz(int tahtaBoyu)
{
    if (TraceMessages)
        Trace.WriteLine("== YZ Motoru Çalıştırılıyor ==");
    _TahtaBoyu = tahtaBoyu;
    _MaxNesil = (int)(Math.Pow(_TahtaBoyu, 2) + 1);
    _DegerCarpan = (int)(Math.Pow(10, (int)(Math.Log10(_TahtaBoyu)) + 2));
    if (TraceMessages)
        Trace.WriteLine("   Tahta Boyu : " + tahtaBoyu);
}

/// <summary>
/// Verilen oyunun oyuncuya göre değerlendirmesini yapar
/// </summary>
```

```
        /// <param name="durum">Oyun tahtasının o anki durumunu içeren
matris</param>
        /// <param name="oyuncu">Durum değerlendirmesini yapacak
oyuncu</param>
        /// <returns>Oyun durumunun oyuncuya göre değeri. Yüksek değer iyi oyun
demektir.</returns>
        private int DurumDegerlendir(AramaAgaci oyun, Oyuncular oyuncu)
        {
            Oyuncular[,] durum = oyun.Oyun;
            //if (TraceMessages)
            //    Trace.Write("Durum Değerlendirmesi (" + oyuncu.ToString() + "): " +
OyunGoster(durum));
            int enb = 0,nDoluOda = 0;

            // Satırları topla
            for (int i = 0; i < _TahtaBoyu; i++)
            {
                int satirDeger = 0;
                for (int j = 0; j < _TahtaBoyu; j++)
                {
                    satirDeger += (int)oyuncu * (int)durum[i, j];
                    if (durum[i, j] != Oyuncular.Hicbiri)
                        nDoluOda++;
                }
                if (Math.Abs(satirDeger) > Math.Abs(enb))
```

```
            enb = satirDeger;
        }

        // Sütunları topla
        for (int j = 0; j < _TahtaBoyu; j++)
        {
            int sutunDeger = 0;
            for (int i = 0; i < _TahtaBoyu; i++)
            {
                sutunDeger += (int)oyuncu * (int)durum[i, j];
            }
            if (Math.Abs(sutunDeger) > Math.Abs(enb))
                enb = sutunDeger;
        }

        // Çaprazları topla
        int caprazDeger = 0;
        for (int i = 0; i < _TahtaBoyu; i++)
        {
            caprazDeger += (int)oyuncu * (int)durum[i, i];
        }
        if (Math.Abs(caprazDeger) > Math.Abs(enb))
            enb = caprazDeger;
```

```
            caprazDeger = 0;
            for (int i = 0; i < _TahtaBoyu; i++)
            {
                caprazDeger += (int)oyuncu * (int)durum[_TahtaBoyu - (1 + i), i];
            }
            if (Math.Abs(caprazDeger) > Math.Abs(enb))
                enb = caprazDeger;

            int retVal = 0;
            if (Math.Abs(enb) != _TahtaBoyu && nDoluOda == _TahtaBoyu *
_TahtaBoyu)    // tahta dolu
                retVal = _BeraberlikDegeri;   // beraberlik
            else
                retVal = enb * _DegerCarpan + (Math.Sign(enb) * (_MaxNesil -
oyun.Nesil));
            return retVal;
        }

        /// <summary>
        /// Oyun durumunun oyun sonunu işaret edip etmediğini belirtir
        /// </summary>
        /// <param name="durum">Oyun tahtasının anlık durumunu belirten Arama
Ağacı düğümü</param>
        /// <returns>true, eğer oyun belirtilen durumda bitiyorsa. false,
bitmiyorsa.</returns>
```

```
public bool OyunBiter(AramaAgaci durum)
{
    int i = DurumDegerlendir(durum, Oyuncular.Bilgisayar);
    if (i == _BeraberlikDegeri)
    {
        durum.OyunDegeri = i;
        return true;
    }
    else if (Math.Abs(i) / _DegerCarpan == _TahtaBoyu)
    {
        durum.OyunDegeri = i;
        return true;
    }
    else
        return false;
}

/// <summary>
/// Verilen andan bir sonraki bütün olası oyunları çıkartır
/// </summary>
/// <param name="a">Oyun anı</param>
/// <param name="oyuncu">Oyun sırası gelen oyuncu</param>
private void OyunBul(AramaAgaci a, Oyuncular oyuncu)
{
```

```
    if (oyuncu != Oyuncular.Hicbiri)
    {
        for (int i = 0; i < _TahtaBoyu; i++)
        {
            for (int j = 0; j < _TahtaBoyu; j++)
            {
                Oyuncular[,] yeniOyun = new Oyuncular[_TahtaBoyu, _TahtaBoyu];
                Array.Copy(a.Oyun, yeniOyun, _TahtaBoyu * _TahtaBoyu);
                if (yeniOyun[i, j] == Oyuncular.Hicbiri) // alan bos
                {
                    yeniOyun[i, j] = oyuncu;    // bulunan oyun
                    a.Oynayan = oyuncu;
                    a.CocukEkle(yeniOyun);
                }
            }
        }
    }
}

private AramaAgaci MaxOyna(AramaAgaci a)
{
    //if (TraceMessages)
    //    Trace.WriteLine("Max Oynuyor.");
    if (OyunBiter(a))
```

```
{
    return a;
}
else
{
    AramaAgaci enIyiOyun = null;
    OyunBul(a, Oyuncular.Bilgisayar);   // agaci buyut
    //if (TraceMessages)
    //    Trace.Write("Nesil : " + a.Nesil + "\nOlası Oyunlar :");
    //if (TraceMessages)
    //    foreach (AramaAgaci b in a.Cocuklar)
    //        Trace.Write(OyunGoster(b.Oyun));
    foreach (AramaAgaci b in a.Cocuklar)
    {
        AramaAgaci oyun = MinOyna(b);
        if (enIyiOyun == null || oyun.OyunDegeri > enIyiOyun.OyunDegeri)
        {
            b.OyunDegeri = oyun.OyunDegeri;
            enIyiOyun = b;
        }
        //if (oyun.OyunDegeri <= ((_TahtaBoyu - 1) * _DegerCarpan) &&
        //    oyun.OyunDegeri > ((_TahtaBoyu - 1) + _DegerCarpan))
        //{
        //    Trace.Write("Oyun bitecek lan olm !!");
```

```
            //}
          }
          Debug.Assert(enIyiOyun != null);
          if (TraceMessages)
            Trace.WriteLine("Max(" + a.Nesil + ") için en iyi oyun : " +
OyunGoster(enIyiOyun.Oyun) + "Değeri : " + enIyiOyun.OyunDegeri);
          return enIyiOyun;
        }
      }

      private AramaAgaci MinOyna(AramaAgaci a)
      {
        //if (TraceMessages)
        //    Trace.WriteLine("Min Oynuyor.");
        if (OyunBiter(a))
        {
          return a;
        }
        else
        {
          AramaAgaci enIyiOyun = null;
          OyunBul(a, Oyuncular.Insan);   // agaci buyut
          //if (TraceMessages)
          //    Trace.Write("Nesil : " + a.Nesil + "\nOlası Oyunlar :");
```

```
            //if (TraceMessages)
            //    foreach (AramaAgaci b in a.Cocuklar)
            //        Trace.Write(OyunGoster(b.Oyun));
            foreach (AramaAgaci b in a.Cocuklar)
            {
                AramaAgaci oyun = MaxOyna(b);
                if (enIyiOyun == null || oyun.OyunDegeri < enIyiOyun.OyunDegeri)
                {
                    b.OyunDegeri = oyun.OyunDegeri;
                    enIyiOyun = b;
                }
            }
            Debug.Assert(enIyiOyun != null);
            if (TraceMessages)
                Trace.WriteLine("Min(" + a.Nesil + ") için en iyi oyun : " +
OyunGoster(enIyiOyun.Oyun) + "Değeri : " + enIyiOyun.OyunDegeri);
            return enIyiOyun;
        }
    }

    private int[] OyundanKoordinata(Oyuncular[,] ilkOyun, Oyuncular[,]
sonrakiOyun)
    {
        for (int i = 0; i < _TahtaBoyu; i++)
```

```
        {
            for (int j = 0; j < _TahtaBoyu; j++)
            {
                if (ilkOyun[i, j] != sonrakiOyun[i, j])
                {
                    if (ilkOyun[i, j] == Oyuncular.Hicbiri)
                        return new int[] { i, j };
                    else
                        throw new
                            InvalidOperationException("OyundanKoordinata: İki oyun
arasında tutarsızlık var");
                }
            }
        }
        throw new
            InvalidOperationException("OyundanKoordinata: İki oyun arasında fark
yok");
    }

    private string OyunGoster(Oyuncular[,] oyun)
    {
        StringBuilder s = new StringBuilder("\n");
        for (int i = 0; i < _TahtaBoyu; i++)
        {
```

```
            s.Append("[\t");
            for (int j = 0; j < _TahtaBoyu; j++)
            {
                if (oyun[i, j] == Oyuncular.Bilgisayar)
                    s.Append("O");
                else if (oyun[i, j] == Oyuncular.Insan)
                    s.Append("X");
                else
                    s.Append(" ");
                s.Append("\t");
            }
            s.Append("]\n");
        }
        return s.ToString();
    }

    public int[] HamleYap(Oyuncular[,] oyun)
    {
        AramaAgaci a = new AramaAgaci(oyun);
        AramaAgaci b = MaxOyna(a), c = b;
        //while (b.Nesil != a.Nesil + 1)
        //    b = b.Baba;
        Oyuncular[,] o = b.Oyun;
        if (OyunBiter(c))
```

```
                RaiseOyunBitti(b.Oynayan);
            return OyundanKoordinata(oyun, o);
        }

        private void RaiseOyunBitti(Oyuncular o)
        {
            OyunBittiDelegate obe = OnOyunBitti;
            if (obe != null)
            {
                obe(this, new OyunBittiEventArgs(o));
            }
        }
    }
}
```

EK-D Örnek AI.CONFIG.XML Dosyası

```
<?xml version="1.0" encoding="utf-8" ?>
<ai-configuration xmlns="http://ajitatif.com/AiConfiguration.xsd">
  <configsets>
    <configset id="törüngey" description="configset for creature törüngey ">
      <modifiers>
        <state value="0.2"/>
        <turns-to-kill value="0.2"/>
        <hp-lost-to value="0.2"/>
        <range value="0.2"/>
        <threat value="0.2"/>
        <target-state-modifier-list>
          <state-modifier state-type="Battle" value="0.5"/>
          <state-modifier state-type="Normal" value="0.25"/>
        </target-state-modifier-list>
      </modifiers>
    </configset>
    <configset id="eser" description="configset for creature eser">
      <modifiers>
        <state value="0.2"/>
        <turns-to-kill value="0.2"/>
        <hp-lost-to value="0.2"/>
        <range value="0.2"/>
```

```
      <threat value="0.2"/>
      <target-state-modifier-list>
        <state-modifier state-type="Battle" value="0.5"/>
        <state-modifier state-type="Normal" value="0.25"/>
      </target-state-modifier-list>
    </modifiers>
  </configset>
 </configsets>
 <training>
  <training-set>
   <party>
    <creatures>
     <creature hit-points="20" armor-class="5"  number-of-attacks="1" pos-x="12" pos-y="10" thac0="10" weapon="Shortbow" is-trainer="true" />
     <creature hit-points="20" armor-class="5" number-of-attacks="1" pos-x="12" pos-y="9" thac0="10" weapon="Shortbow" />
    </creatures>
   </party>
   <party>
    <creatures>
     <creature hit-points="20" armor-class="5" number-of-attacks="1" pos-x="9" pos-y="12" thac0="10" weapon="Shortbow" />
     <creature hit-points="20" armor-class="5" number-of-attacks="1" pos-x="9" pos-y="9" thac0="10" weapon="Shortbow" />
    </creatures>
```

```
      </party>
    </training-set>
  </training>
</ai-configuration>
```

EK-E RPG Yapay Zekâ Kodu

```
using System;
using System.Collections.Generic;
using System.Text;
using Rpg.Entity;
using System.Diagnostics;
using Rpg.Actions;
using Rpg.Entity.CreatureStates;
using Rpg.Rules;
using System.Xml;
using System.IO;
using System.Reflection;
using System.Globalization;
using Rpg.Common;

namespace Rpg.Ai
{
    public class AiPlayer : PlayerBase
    {
        #region Score Modifier Coefficients

        public double
            kState = 0.2,
```

```
    kTurnsToKill = 0.2,
    kHpLostTo = 0.2,
    kRange = 0.2,
    kThreat = 0.2;

#endregion

#region State Modifiers

private readonly Dictionary<Type, double> _stateModifiers;

#endregion

private static string DefaultConfigFilename = "Ai.config.xml";
private Realm _realm;
private Creature _creatureControlled;
private List<Creature> _hostileCreatures;
private List<Creature> _friendlyCreatures;
private List<AttackOrderListEntry>  _attackOrderList;

public List<AttackOrderListEntry> AttackOrderList
{
    get { return _attackOrderList; }
}
```

```
        public Creature CreatureControlled { get { return _creatureControlled; } }

        public AiPlayer(Creature creatureToControl)
        {
            _realm = Realm.GetInstance();
            _creatureControlled = creatureToControl;
            _creatureControlled.Stats.Player = this;

            _attackOrderList = new List<AttackOrderListEntry>();
            _stateModifiers = new Dictionary<Type, double>();

            InitializeStateModifiers();
            ReadConfigFromXml(DefaultConfigFilename,
creatureToControl.Stats.Name);
            UpdateCreatureAlignments();
        }

        public void UpdateCreatureAlignments()
        {
            _hostileCreatures = new List<Creature>();
            _friendlyCreatures = new List<Creature>();
            foreach (Creature c in _creatureControlled.Stats.Position.Board.Actors)
            {
```

```
            if (c != _creatureControlled && !c.IsDead())
            {
                if (c.GetAlignmentAgainst(_creatureControlled) ==
GetCreatureTypeParam.Hostile)
                {
                    _hostileCreatures.Add(c);
                }
                else
                {
                    _friendlyCreatures.Add(c);
                }
                AttackOrderListEntry aole = GetAttackListEntry(c);
                if (aole == null && _hostileCreatures.Contains(c))
                {
                    aole = new AttackOrderListEntry();
                    aole.TargetCreature = c;
                    _attackOrderList.Add(aole);
                }
            }
        }
    }

    private void InitializeStateModifiers()
    {
```

```
        _stateModifiers.Add(typeof(Battle), 0.5);
        _stateModifiers.Add(typeof(Normal), 0.25);
    }

    private void UpdateAttackOrderList()
    {
        foreach (AttackOrderListEntry aole in _attackOrderList)
        {
            Type t = aole.TargetCreature.Stats.CreatureState.GetType();
            if (_stateModifiers.ContainsKey(t))
            {
                CalculateAttackOrderListEntry(aole);

                double stateModifier = _stateModifiers[t];
                double ttkModifier, rngModifier;
                if (kTurnsToKill == 0)
                {
                    ttkModifier = 0;
                }
                else
                {
                    ttkModifier = kTurnsToKill * (1.0 / Math.Max(1,
aole.NumberOfTurnsToKill));
                }
```

```
            if (kRange == 0)
            {
                rngModifier = 0;
            }
            else
            {
                rngModifier = kRange * (1.0 / Math.Max(1, aole.RangeModifier));
            }

            aole.Score =
                kState * stateModifier +
                ttkModifier +
                kHpLostTo * aole.HitPointsLostTo +
                rngModifier +
                kThreat * aole.ThreatModifier
            ;
        }
    }
    _attackOrderList.Sort();
}

private void CalculateAttackOrderListEntry(AttackOrderListEntry aole)
{
```

```
        aole.RangeModifier =
            Math.Max(0,
            _creatureControlled.DistanceToCreature(aole.TargetCreature) -
_creatureControlled.Weapon.Range);


        if (_creatureControlled.Weapon.Damage.MaxRoll *
_creatureControlled.Stats.NumberOfAttacks == 0)
        {
            aole.NumberOfTurnsToKill = int.MaxValue;
        }
        else
        {
            aole.NumberOfTurnsToKill =
                aole.TargetCreature.Stats.HitPoints /
                (_creatureControlled.Weapon.Damage.MaxRoll *
_creatureControlled.Stats.NumberOfAttacks);
        }
        if (aole.LastDamageTaken == 0)
        {
            aole.ThreatModifier = 0;
        }
        else
        {
            aole.ThreatModifier = _creatureControlled.Stats.HitPoints /
aole.LastDamageTaken;
```

```
        }
    }

    public override void MakeMove()
    {
        ModifyCoefficients();
        UpdateAttackOrderList();
        if (_hostileCreatures.Count > 0)
        {
            //Creature target =
_creatureControlled.GetNearestCreature(GetCreatureTypeParam.Hostile);
            int targetIndex = _attackOrderList.Count - 1;
            Creature target = _attackOrderList[targetIndex].TargetCreature;  // get the
first in the list
            while (target.IsDead())
            {
                if (targetIndex > 0)
                {
                    target = _attackOrderList[--targetIndex].TargetCreature;
                    continue;
                }
                break;
            }
            Debug.Assert(target != null);
```

```
            if (_creatureControlled.DistanceToCreature(target) >=
_creatureControlled.Weapon.Range)
            {
                MoveCreatureTowards(target.Stats.Position);
            }
            else
            {
                Attack.CreatureToCreature(_creatureControlled, target);
            }
        }
        UpdateCreatureAlignments();
        UpdateAttackOrderList();
    }

    private void ModifyCoefficients()
    {
        // get primary modifier
        double selfParty = (double)(_creatureControlled.Stats.HitPoints) /
_creatureControlled.Stats.MaxHitPoints,
            otherParty = 0;
        foreach (Creature creature in _friendlyCreatures)
        {
            selfParty += (double)(creature.Stats.HitPoints) /
creature.Stats.MaxHitPoints;
        }
```

```
            selfParty /= _friendlyCreatures.Count + 1;

            foreach (Creature creature in _hostileCreatures)
            {
                otherParty += (double)(creature.Stats.HitPoints) /
creature.Stats.MaxHitPoints;
            }

            otherParty /= _hostileCreatures.Count;

            double primaryModifier = selfParty / otherParty;
        }

        private void MoveCreatureTowards(BoardPosition boardPosition)
        {
            BoardPosition currentPos = _creatureControlled.Stats.Position;
            if (boardPosition.Board == currentPos.Board)
            {
                int signY = -1 * Math.Sign(currentPos.Point.Y - boardPosition.Point.Y);
                int signX = -1 * Math.Sign(currentPos.Point.X - boardPosition.Point.X);
                if (boardPosition.Point.X == currentPos.Point.X)
                {
                    _creatureControlled.MoveBy(0, signY);
```

```
        }
        else if (boardPosition.Point.Y == currentPos.Point.Y)
        {
            _creatureControlled.MoveBy(signX, 0);
        }
        else
        {
            // move through either X or Y, selected by random
            int randomNumber = _realm.Randomizer.Next(0, 2);
            if (randomNumber == 0)
            {
                signX = 0;
            }
            else
            {
                signY = 0;
            }
            _creatureControlled.MoveBy(signX, signY);
        }
    }
}

private AttackOrderListEntry GetAttackListEntry(Creature creature)
{
```

```
    AttackOrderListEntry aole = null;
    foreach (AttackOrderListEntry a in _attackOrderList)
    {
        if (a.TargetCreature == creature)
        {
            aole = a;
            break;
        }
    }
    return aole;
}

public override void AfterTakeDamage(Creature attacker, int damage)
{
    base.AfterTakeDamage(attacker, damage);
    // update the Attack Order List
    AttackOrderListEntry aole = GetAttackListEntry(attacker);
    if (aole == null)
    {
        aole = new AttackOrderListEntry();
        aole.TargetCreature = attacker;
        _attackOrderList.Add(aole);
    }
    aole.HitPointsLostTo += damage;
```

```
            aole.LastDamageTaken = damage;
        }

        private void ReadConfigFromXml(string fileName, string configSetId)
        {
            _stateModifiers.Clear();
            NumberFormatInfo formatter = new NumberFormatInfo();
            formatter.NegativeSign = "-";
            formatter.PositiveSign = "+";
            formatter.NumberDecimalSeparator = ".";

            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load(fileName);
            XmlNamespaceManager nsManager = new
XmlNamespaceManager(xmlDoc.NameTable);
            nsManager.AddNamespace("ai", "http://ajitatif.com/AiConfiguration.xsd");

            XmlNode configSet =
                xmlDoc.SelectSingleNode(
                    string.Format(@"//ai:configset[@id=""{0}""]", configSetId.ToLower())
                    , nsManager);
            if (configSet == null)
            {
```

```
            if (string.Equals(configSetId, "default",
StringComparison.InvariantCultureIgnoreCase))
            {
                double k1 = _realm.Randomizer.NextDouble(),
                    k2 = _realm.Randomizer.NextDouble(),
                    k3 = _realm.Randomizer.NextDouble(),
                    k4 = _realm.Randomizer.NextDouble(),
                    k5 = _realm.Randomizer.NextDouble(),
                    kTotal = k1 + k2 + k3 + k4 + k5;
                kHpLostTo = k1 / kTotal;
                kRange = k2 / kTotal;
                kState = k3 / kTotal;
                kThreat = k4 / kTotal;
                kTurnsToKill = k5 / kTotal;
                _stateModifiers.Add(typeof(Battle), 0.5);
                _stateModifiers.Add(typeof(Normal), 0.25);
                return;
            }
            else
            {
                ReadConfigFromXml();
                return;
            }
        }
```

```
        try
        {
            kHpLostTo =
Convert.ToDouble(configSet.SelectSingleNode(@"./ai:modifiers/ai:hp-lost-
to/@value", nsManager).Value, formatter);
            kRange =
Convert.ToDouble(configSet.SelectSingleNode(@"./ai:modifiers/ai:range/@value",
nsManager).Value, formatter);
            kState =
Convert.ToDouble(configSet.SelectSingleNode(@"./ai:modifiers/ai:state/@value",
nsManager).Value, formatter);
            kThreat =
Convert.ToDouble(configSet.SelectSingleNode(@"./ai:modifiers/ai:threat/@value",
nsManager).Value, formatter);
            kTurnsToKill =
Convert.ToDouble(configSet.SelectSingleNode(@"./ai:modifiers/ai:turns-to-
kill/@value", nsManager).Value, formatter);

            XmlNode targetStateModifierListElement =
configSet.SelectSingleNode(@"./ai:modifiers/ai:target-state-modifier-list",
nsManager);

            Assembly entityAssembly = Assembly.GetAssembly(typeof(Battle));

            foreach (XmlNode elem in targetStateModifierListElement.ChildNodes)
            {
                string stateTypeString = elem.Attributes["state-type"].Value;
                Type stateType = Type.GetType(stateTypeString);
```

```
                if (stateType == null)
                {
                    stateType =
Type.GetType(string.Format("Rpg.Entity.CreatureStates.{0}, {1}", stateTypeString,
entityAssembly.FullName), true, true);
                }
                double stateValue = Convert.ToDouble(elem.Attributes["value"].Value,
formatter);
                _stateModifiers.Add(stateType, stateValue);
            }
        }
        catch (Exception ex)
        {
            throw new Exception("Unable to parse the config file", ex);
        }
    }

    private void ReadConfigFromXml()
    {
        ReadConfigFromXml(DefaultConfigFilename, "default");
    }
  }
}
```

ÖZGEÇMİŞ

1983 yılında Tekirdağ İli'nin Çorlu ilçesinde doğdu. İlk ve orta öğrenimini Ankara'da, lise öğrenimini ise İstanbul'da tamamladı. 2001 yılında girdiği Kocaeli Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nden 2005 yılı Haziran ayında mezun oldu ve aynı yılın eylül ayında Kocaeli Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Bölümü'ne yüksek lisans eğitimine kabul edildi. 2006 yılı Ekim ayı itibariyle hakia.com'un İstanbul İrtibat Ofisi'nde yazılım mühendisi olarak çalışmaktadır ve bekârdır.