ALGORİTMİK OYUN KURAMI

Dr. Öğr. Üyesi Erol KINA

İKSAD Publishing House

ALGORİTMİK OYUN KURAMI

Dr. Öğr. Üyesi Erol KINA

İKSAD
Publishing House

(The Licence Number of Publicator: 2014/31220)
TURKEY TR: +90 342 606 06 75
USA: +1 631 685 0 853
E mail: iksadyayinevi@gmail.com
www.iksadyayinevi.com
It is responsibility of the author to abide by the publishing ethics rules.
Iksad Publications – 2023©
ISBN: 978-625-367-188-4
Cover Design: Erol KINA
Editör: Dr. Öğr. Üyesi Emre BİÇEK
July / 2023
Ankara / Türkiye
Size = 16 x 24 cm

ÖNSÖZ

Sevgili okuyucular,

Sizi, algoritmik oyun kuramıyla heyecan verici bir yolculuğa davet ediyorum! Bu kitap, algoritmik oyun kuramının temellerini, uygulamalarını ve stratejilerini anlamak için bir rehber niteliğindedir. Oyun teorisi, çeşitli disiplinlerde büyük ilgi uyandıran bir alan haline gelmiştir ve algoritmik oyun kuramı, bu alana matematiksel ve algoritmik bir yaklaşım getiren önemli bir bileşendir. Son 25 yılda, bilgisayar bilimi ve ekonomi arasında yoğun bir etkileşim söz konusu olmuş ve bu etkileşim algoritmik oyun kuramı olarak adlandırılan yeni bir alanın ortaya çıkmasına yol açmıştır. Bu kitap, "Algoritmik Oyun Kuramı" adlı yüksek lisans dersim için kaynak niteliğindedir. 2022 yılından beri Van Yüzüncü Yıl Üniversitesinde bu dersi vermekteyim.

Algoritmik oyun kuramı, karmaşık sistemlerde karar verme süreçlerini analiz etmek, stratejiler oluşturmak ve çıkarlarını en üst düzeye çıkarmak isteyen oyuncuların davranışlarını anlamak için güçlü bir araçtır. Bu kitapta, algoritmik oyun kuramının temel kavramlarını, matematiksel modellerini ve analiz yöntemlerini adım adım keşfedeceksiniz. Ayrıca, pratik uygulamalara odaklanarak, gerçek dünya problemlerini çözmede nasıl kullanılabileceğini de göreceksiniz.

Kitabım, hem teorik bir temel sağlayan kapsamlı bir kaynak olmayı hem de okuyucuların algoritmik oyun kuramını somutlaştırmalarına yardımcı olacak pratik örnekler sunmayı amaçlamaktadır. Her bölümde, algoritmaların kullanımını anlatan adım adım örnekler ve açıklayıcı şekillerle birlikte öğrenmeyi destekleyici bir yaklaşım sunarak konuların daha anlaşılır olmasını sağlamaya çalıştım.

Algoritmik oyun kuramı, ekonomi, bilgisayar bilimi, yapay zekâ, optimizasyon, dağıtık sistemler ve ağlar gibi daha birçok alanda uygulamaları olan çok disiplinli bir konudur. Dolayısıyla, bu kitap, bu alanlarda çalışan öğrenciler, araştırmacılar ve uygulayıcılar için değerli bir kaynak olmayı hedeflemektedir.

Bu kitabın amacı, algoritmik oyun kuramı konusunda okuyuculara genel bir anlayış sağlamak ve temel kavramları açıklamaktır. Ayrıca, okuyucuların algoritmik oyun kuramı alanında daha ileri düzeyde çalışmalar yapmalarına

olanak tanıyacak bir temel sağlamayı hedeflemektedir. Kitap, oyun kuramının matematiksel temellerini ele alarak, oyuncuların stratejik etkileşimlerini ve karar verme süreçlerini analiz etmeyi amaçlamaktadır.

Kitabın kapsamı, algoritmik oyun kuramının çeşitli yönlerini içermektedir. Oyun kuramının temel kavramlarının yanı sıra, oyuncuların stratejik seçimleri ve oyunların matematiksel modelleri gibi konulara odaklanmaktadır. Nash dengesi gibi önemli kavramlar detaylı bir şekilde incelenirken, oyuncuların optimal stratejilerini belirleme ve kazançlar üzerindeki etkileri üzerinde durulmaktadır. Kitap, çoklu oyunculu oyunlar ve bu oyunların stratejik etkileşimlerini ele almaktadır. Bu kitap, algoritmik oyun kuramıyla ilgilenen akademisyenler, öğrenciler ve araştırmacılar için önemli bir kaynak niteliği taşımaktadır. Okuyucular, kitap sayesinde algoritmik oyun kuramının temel kavramlarını, yöntemlerini ve uygulamalarını anlayarak konuya daha derinlemesine hâkim olabileceklerdir. Kitapta verilen Kodların tamamı açık adresi belirtilen internet tabanlı bir depolama servisi olan GitHub platformuna yüklenmiştir. GitHub platformuna yüklenen örnek kodlara ulaşmak için aşağıdaki karekodu taratabilirsiniz. Kitapta bulunan bütün Python kodlamalarında, Anaconda.Navigator versiyon: 1.10.0 ve Jupyter Notebook versiyon: 6.1.4 kullanılmıştır.

Saygılarımla,

Dr. Öğr. Üyesi Erol KINA

https://github.com/erolkina/Algotirmik_Oyun_Kurami/commit/d91d894b461c48b9af6c71185213bfc67a7dd5e0

Bu kitabı, daima sevgi ve destekleri ile yanımda olan güzel annem Gönül KINA'ya, yakışıklı babam Uzm. Dr. Celil KINA'ya, prenses kızım Gönül Su KINA'ya ve yakışıklı oğlum Rüzgar Ege KINA'ya armağan ediyorum.

Sizin sevginiz ve desteğiniz bu kitabın her sayfasında yer alacak.

Dr. Öğr. Üyesi Erol KINA

İÇİNDEKİLER

BİRİNCİ BÖLÜM

1. GİRİŞ

Oyun teorisi, insanların sosyal ve ekonomik kararlarını hesaplamalar ve analizler yoluyla ele alan bir bilimsel teoridir. Bu teori, hayatın her anının bir oyun olduğunu unutarak, insanların karşılıklı ilişkilerde ve kişisel çıkarların söz konusu olduğu her durumda karar verme süreçlerini incelemeyi amaçlar. Tarihe baktığımızda, çeşitli örneklerle oyun teorisi karşımıza çıkar. Örneğin, Atinalılar ve Böotyalılar arasında MÖ 424 yılında gerçekleşen ünlü Delium Muharebesi, oyun teorisinin ilk örneklerinden biridir. Platon'un yazılarında bu muharebeyle ilgili bir senaryo kurgulanmıştır: Savunan bir Böotyalı asker olduğunuzu düşünelim. Size karşı üstün bir ordu geliyor ve siz en ön sıradasınız. Savunmanızın başarılı olması durumunda bile kayıpların kaçınılmaz olduğunu biliyorsunuz. Ayrıca en ön sırada olduğunuz için ölme olasılığınız çok yüksektir. Savunmanız başarısız olursa zaten öleceksiniz. Bu durumda, karşı askerlere baktığınızda, kaçmanın en mantıklı çözüm olduğunu düşünüyorsunuz. Ancak burada bir sorun ortaya çıkıyor: Tüm askerlerin aynı mantığı izlediğini varsayalım. Kaçmanın en mantıklı çözüm olduğunu fark ettiğimizde, Böotyalılar birkaç cephede iyi savaşçılara sahip olmalarına rağmen topraklarının işgalini engelleyememişlerdir. Platon'un dikkat çektiği nokta, bu senaryonun oyun teorisi açısından tarihteki ilk örneklerinden biri olmasıdır. Bu durum sadece askerlerin bakış açısından ele alınsa da genel olarak savaş, kelime anlamıyla bir "oyun" niteliği taşıyabilir. En azından belirli kuralları olan bir tarafın diğer tarafa üstünlük sağlamaya çalıştığı ve sonunda belirli ödüllerin kazanıldığı bir "oyun" söz konusudur.

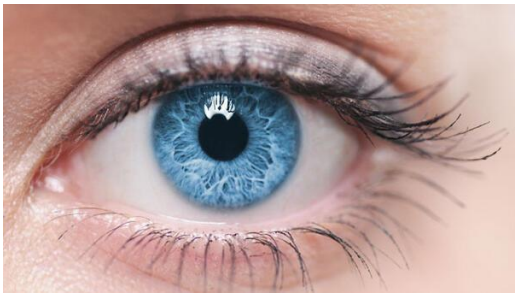
Şimdi 1944 yılına hızlı bir sıçrama yapalım. Macar asıllı Amerikalı matematikçi John von Neumann, "Theory of Games and Economic Behavior" (Oyun Teorisi ve Ekonomik Davranış) isimli önemli bir kitap yayınlamıştı. Bu kitapta von Neumann, Zero-sum oyunları olarak adlandırılan ve bir kişinin kazancının diğerinin kaybı anlamına geldiği oyunlar için, her iki tarafın da yararına olan çözümler bulmanın yöntemlerini ele almıştı. Ayrıca kitapta oyun teorisinin ekonomiden siyasete, spordan bilime kadar nasıl uygulanabileceği de anlatılmıştı (Neumann ve Morgenstern, 1947).

Oyun teorisi, 1994 yılında Nobel ödülü alan ve tarihinin en önemli dehalarından biri olarak kabul edilen John Nash tarafından yaygınlaştırılmış ve ekonomi alanında büyük ilgi görmüştür (Kuhn ve ark., 1996).

Oyun teorisi, birden fazla kişinin bulunduğu durumlarda her bireyin kararlarının diğerlerini etkilediği her türlü etkileşimli durumu analiz etmek için kullanılan bir bilimsel teoridir. Oyun teorisi, arkadaşlık ilişkilerinden alışverişe ve hatta evlilik gibi çeşitli alanlarda geçerlidir. Bu "oyun" kavramının temel unsurları bulunmaktadır. İlk olarak, en az iki kişi olmalıdır. İkinci olarak, kişiler

arasında etkileşim olmalıdır. Üçüncü olarak, bir ödül veya sonuç bulunmalıdır. Dördüncü olarak, tarafların mantıklı ve rasyonel davranması beklenir. Rasyonel davranmak yaptığımız tercihlerin mantıksal bir tutarlılığa sahip olması anlamına gelir (Dekel ve Gul, 1997). Bir kişinin çay ve kola seçeneklerinde sürekli olarak çay içmeyi tercih ettiğini varsayalım. Çay, kola ve kahve seçeneklerinde çay ya da kahve seçeneklerinden birisini tercih edeceğini söyleyebilmeliyiz. Son olarak, tarafların kendi çıkarları doğrultusunda hareket ettikleri varsayılır. Oyuncular, stratejiler, kazançlar ve kurallar rasyonellik çerçevesinde ortak bilgiyi oluştururlar. Bilgi bir kişinin A konusunu bilmesi, karşılıklı bilgi her birimizin A bilgisini bilmesi, ortak bilgi ise her birimiz her birimizin A bilgisini bildiğini bilmesi olarak ifade edilebilir. Rasyonelliği ortak bilgiyle anlatmaya çalışırsak, her oyuncu diğer oyuncuların rasyonel olduklarını bildiklerini bilir diyebiliriz. Ortak bilgi ile alakalı bir problem çözelim.

Problem: Konuşmanın yasak olduğu bir adada n adet kişinin göz rengi mavidir ve adada yaşan diğer kişilerin göz renkleri siyahtır. Göz renginin mavi olduğunu anlayan kişi ertesi gün adayı terk etmek zorundadır. Adaya dürüst, doğru konuşan yabancı bir kişi gelip bir duyuru yapar. Duyuruda şunu söyler: “Bu adada yaşayan en az bir kişinin göz rengi mavidir” (Ortak Bilgi). Gözü mavi renkli olan kişiler adayı ne zaman terk ederler?



Çözüm:

n = 1; Gözü mavi renkli olan kişi herkesin gözünün mavi olduğunu görüp ertesi gün adayı terk eder.

n > 1; n en az 2 olsun. O halde her mavi gözlü kişi şöyle düşünür: "Eğer ben mavi gözlü değilsem kabilede sadece n-1 mavi gözlü olabilir. Bu durumda n-1. gün tüm n-1 mavi gözlü adayı terk edecekti. Fakat n-1'inci günün akşamına

gelindiğinde kimse adayı terk etmeyince tüm mavi gözlüler kendilerinin de mavi gözlü olduğunu anlarlar. Nitekim ertesi gün (yani n'inci gün) hepsi birden adayı terk ederler.

n* = *n; n gün sonunda bütün mavi gözlüler adayı terk etmiş olur.

Bu çözümde en az iki kişinin olduğu, kişiler arasında etkileşimin sınırlı olduğu, bir sonuca varıldığı, tarafların mantıksal ve rasyonel davrandığını söyleyebiliriz. Adaya gelen yabancı kişinin ise duyuruyu yaparak ortak bilginin habercisi olduğu görülmektedir. Ortak bilgi sayesinde kişiler kendi göz renklerinin mavi olduğunu anlamış oldular.

Günlük hayatta karar verirken matematiksel düşünce, stratejik planlama ve olası senaryoları değerlendirme becerisi, bizi daha bilinçli ve analitik kararlar almaya yönlendirir. Matematiksel yaklaşımlar, günlük yaşamımızın her alanında etkili bir şekilde kullanılabilecek değerli araçlar sunar. Mahkûm İkilemi, oyun teorisi açısından ilgi çekici bir durumu temsil etmektedir ve çeşitli alanlarda incelenmiştir. Bu durum, iki oyuncunun kendi çıkarlarını gözeterek karar vermesini ve bu kararların birbirlerinin kararlarına bağımlı olduğu bir etkileşimi içermektedir. Mahkûm İkilemi, stratejik düşünceyi gerektiren bir durumu ifade etmekte olup oyuncuların kararlarını verirken birbirlerinin stratejilerini dikkate almalarını gerektirmektedir (Bhaumik, Roy ve Weber, 2020).

Problem: Polis anlaşmasına göre, eğer bir mahkûm diğerini ele verirse, sadece bir yıl hapis cezası alacak ve serbest bırakılacaktır ve diğer mahkûm 10 yıl ceza alacaktır. Ancak her ikisi de susarsa, polisin yeterli kanıt olmadığı için her biri maksimum bir yıl ceza alabilir. Bununla birlikte, her ikisi de birbirini ele verirse, her biri beş yıl ceza alır.

Öncelikle, başlangıçta iki suçlunun sessiz kalmalarının en iyi strateji olduğu gibi bir izlenim ortaya çıkmaktadır. Bu durumda, her iki suçlu en düşük ceza olan iki yıl hapis cezasıyla sonuçlanan bir çıkar sağlayabilir. Ancak, bu durumda karşı tarafın ne yapacağı belirsizdir, bu nedenle suçlu şu bir muhasebe yapar: Eğer sessiz kalırsam ve diğer suçlu beni ele verirse, ben on yıl hapis cezası alırım. Bu nedenle, her zaman bireysel çıkarlarını en üst düzeyde koruma hedefiyle, karşı tarafın kararlarından bağımsız olarak en optimal stratejiyi seçmek en mantıklı seçenek olarak değerlendirilir. Bu durum "Nash dengesi" veya "baskın strateji" olarak adlandırılmaktadır. Neredeyse her zaman kişisel çıkarlar, grubun çıkarları önünde önceliklidir. Ancak, bu stratejinin her zaman

Nash dengesiyle uyumlu olması gerektiğini unutmamak önemlidir (Elgazzar, 2019). Bu teoriyi her yerde görürüz, Apple ve Samsung gibi dev şirketler arasındaki rekabetten, Rusya ve Amerika gibi büyük çatışmalara kadar birçok alanda etkisini görebiliriz.

John Nash tarafından ortaya konulan teori, oyuncuların fayda grafiğindeki optimum noktayı bulmaları ve stratejilerini buna göre belirlemeleri gerektiğini ifade etmektedir. Bu teori, beklenenden daha büyük bir etkiye sahip olmuş ve modern dünyanın şekillenmesinde önemli bir rol oynamıştır (Ye ve ark., 2023). İnsanlar veya şirketler, kendi çıkarlarını gözeterek ve bunları en üst düzeye çıkarmaya çalışarak hareket etmektedir. Ancak, iş birliği yapılmadığında, ortaklık aranmadığında veya benzer çıkarlara sahip taraflar arasında bir anlayış sağlanmadığında, sonuçların uzun vadede herkesin zararına olduğu gözlenmektedir. Bu nedenle, dikkat edildiğinde, piyasada büyük rakipler gibi görünen şirketlerin aslında sürekli olarak birlikte hareket ettikleri, birbirlerinin çıkarlarını gözetledikleri fark edilebilir. Bu durum hem onların hem de genel olarak tüm tarafların çıkarları için gerekli bir dengeyi sağlamaktadır.

Oyun teorisi, genel anlamıyla hayatı kapsayıcı bir şekilde özetleyebilmektedir ve biyoloji gibi diğer disiplinlere de uygulanabilir (Mcnamara ve Leimar, 2020). Biyologlar, oyun teorisini biyolojiye uyguladıklarında, biyolojik fedakârlık kavramıyla karşılaşmaktadırlar. Bu kavram, bir organizmanın türünün devamı için kendi çıkarlarına zarar verebilecek durumlarda bile fedakârlık yapması üzerine odaklanır. Örneğin, bir kuş düşünelim. Kuş, bir avcı yaklaştığında diğer kuşları uyarmak için büyük bir çaba sarf ederek avcıya dikkat çeker. Bu davranışıyla kendisini tehlikeye atarken, diğer kuşların da benzer şekilde tepki vereceğini ve hayatta kalma şanslarının artacağını bilir. İnsanlara gelindiğinde, bu denge sonucunda bazı türlerin yok olmasına şahit olunmuştur. Ancak, insan türü doğal olarak hayatta kalmayı başarmıştır, bu da binlerce yıllık doğru stratejilerin bir sonucudur.

Kuş örneğinde olduğu gibi, ideal şartlarda herkes üzerine düşeni yapmaktadır. Ancak insan ilişkilerini ele aldığımızda, maalesef tahmin edilemezlik büyük bir sorun yaratmaktadır. Çöpü yere atmaktan bahsettiğiniz gibi basit bir örnekte bile, Nash dengesini yerle bir ettiğimizi söyleyebiliriz. Kendi evimizin önünü süpürmek ve çöpleri atmamak hem kendimiz hem de genel olarak herkesin faydasına olmasına rağmen, çöp atanlardan

kurtulamıyoruz. Dengeyi sürekli bozanlar ve "herkes atıyor kardeşim" diyenlerin dengenin böyle olduğunu düşünmesi sorun oluşturuyor. Maalesef, bu nedenle kendi hayatınızda da bir oyun teorisyeni gibi hareket etmeniz gerekiyor. Herhangi bir tartışmaya veya zorlu bir duruma girdiğinizde, şu soruları sorabilirsiniz: Burada herkes akılcı davranıyor mu? Herkes, herkesin faydasına olacak bir denge kurabiliyor mu? Farklılıkları bir kenara bırakıp, herkesin faydasına olacak bir denge kurulabiliyor mu? Herkes sadece kendi çıkarlarını mı gözetiyor? Oyunun kurallarını anlamışlar mı? Bu tür parametrelerin karşılanmadığını gördüğünüzde, yaklaşımınızı değiştirmeniz gerekebilir.

Örneğin, trafikte güvenli ve düzenli bir şekilde seyahat etmek için trafik kurallarına uymak önemlidir. Her sürücünün kendi çıkarlarını düşünerek trafik kurallarını ihlal etmesi, genel olarak trafik akışını bozacak ve kazalara yol açabilecektir. Bu durumda, bir oyuncu olarak herkesin faydasına olacak bir dengeyi sağlamak için Nash dengesini gözetmek gerekmektedir. Ancak, trafikte bazı sürücülerin kurallara uymamaları veya kural ihlali yapan diğer sürücülerin varlığını bahane edenlerin sayısı arttıkça, bu denge bozulabilir. Örneğin, bir sürücü trafik kurallarına uymaya devam ederken, diğer sürücülerin kurallara uymaması nedeniyle sık sık risk altına girebilir ve dezavantajlı duruma düşebilir. Bu durumda, her sürücünün sadece kendi çıkarlarını düşünerek kural ihlali yapması, genel olarak trafikte düzensizliğe ve kazalara neden olabilir. Bu noktada, bireysel olarak kendi çıkarlarını korumak yerine, tüm sürücülerin faydasına olacak bir dengeyi sağlamak için iş birliği ve uyum önem kazanır. Tüm sürücülerin kurallara uyması, güvenli ve verimli bir trafik akışını sağlar. Trafik örneği üzerinden görüyoruz ki, oyun teorisi prensiplerini uygulayarak herkesin faydasına olacak bir dengeyi sağlamak önemlidir. Her oyuncunun bireysel çıkarlarını gözetirken, aynı zamanda diğer oyuncuların davranışlarını ve kararlarını da dikkate alarak hareket etmek, uzun vadede daha sürdürülebilir ve adil sonuçlar elde etmemizi sağlar.

1.1. Oyun Kuramının Tanımı ve Önemi

İnsan sistemleri, çeşitli kurumlar, organizasyonlar ve sistemler aracılığıyla birbirleriyle etkileşim halinde olan karmaşık yapılar olarak

düşünülebilir. Bu sistemler, çeşitli paydaşların çıkarlarının çakışması, rekabet etme, iş birliği yapma ve birbirlerine bağımlılık gibi özelliklerle karakterizedir.

Örneğin, bir işletme içindeki çalışanlar arasında rekabet ve iş birliği ilişkileri, firmaların rekabeti ve iş birliği içinde olduğu pazarlar, hükümetin politika yapma süreçleri veya sivil toplum kuruluşlarının toplumsal etkileşimleri gibi durumlar bu sistemlere örnek olarak verilebilir.

Bu sistemlerde, oyuncuların (aktörlerin) kararları ve eylemleri, diğer oyuncuların kararları ve eylemleriyle etkileşim halindedir. Oyun kuramı, bu tür stratejik etkileşimleri matematiksel bir modelleme aracı olarak kullanır. Özellikle, bir grup firmadan oluşan bir sistemin rekabetçi davranışlarını analiz etmek için oyun kuramını kullanmak akıllıcadır. Oyun kuramı, oyuncuların tercihlerini, eylem seçeneklerini ve bu eylemlerin sonuçlarına ilişkin tercihlerini analiz eder. Bir firmanın alabileceği eylemler, diğer firmaların aldığı eylemlere bağlı olarak farklı sonuçlara yol açabilir. Örneğin, bir firmanın fiyatını düşürmesi, rakip firmaların karlılığını etkileyebilir. Oyun kuramı, bu stratejik etkileşimleri matematiksel olarak modelleyerek, oyuncuların optimal stratejilerini belirlemelerine ve potansiyel sonuçları tahmin etmelerine yardımcı olur ayrıca, stratejik etkileşimleri anlamak ve analiz etmek için kullanılan bir araçtır. Karar vericiler, oyuncuların stratejilerini ve potansiyel tepkilerini değerlendirerek en iyi eylem stratejilerini belirleyebilirler. Bu analiz, oyuncuların kazançlarını maksimize etme veya kayıplarını minimize etme amacına hizmet eder (Grüne-Yanoff ve Lehtinen, 2012).

"Rasyonel, zeki" yöneticilere sahip bir firmanın "Diğer firmalar ne yapacak?" sorusuna verebileceği cevaplar üzerinde kısıtlamalar getirmek, kararlarının tamamının birbirleriyle ilişkili olduğu durumlarda kullanılan bir yaklaşımdır (Moorthy, 1985). Kararların ve kalitenin, bu kararları etkileyen temel parametreler olduğu ve tüm oyuncuların kararlarının birbirleriyle ilişkili olduğu söylenebilir (Pereira, 2014). Oyuncular arasındaki bu etkileşimi incelemek için sunulan model, oyuncuların eylemlerini çalışmak ve tahmin etmek için en yaygın kullanılan teorilerden biri olarak kabul edilen rasyonel tercih teorisine dayanacaktır (Askari ve ark., 2019). Bu teori, oyuncuların diğer oyuncuların stratejilerini tahmin etmelerine olanak tanır (Colman, 2003).

Oyun kuramının önemi, birçok alanda uygulama alanı bulmasından kaynaklanmaktadır. Ekonomi, yapay zekâ, işletme, politika, sosyoloji, psikoloji, fizik ve biyoloji gibi disiplinlerde yaygın olarak kullanılmaktadır

(Shoham, 2008; Samuelson, 2016; Hauert ve Szabó, 2005; McNamara, 2022; Elkind ve Leyton-Brown, 2010; Solomon, 1999; Stolz, 2023). Örneğin, iş dünyasında, rekabet stratejilerini belirlemek ve piyasa koşullarını analiz etmek için oyun kuramı kullanılır. Politika alanında, oyuncular arasındaki etkileşimleri ve politika kararlarının sonuçlarını anlamak için kullanılır. Sosyal bilimlerde, insan davranışını anlamak ve toplumsal etkileşimleri modellemek için kullanılır. Bütün bunları karar verme süreçlerini analiz etmek için matematiksel modeller ve kavramlar kullanarak gerçekleştirir. Oyun ağaçları, oyun matrisleri, Nash dengesi, Pareto etkinliği ve koalisyon teorisi gibi kavramlar, oyuncuların stratejilerini ve potansiyel sonuçlarını anlamak için kullanılır. Bu matematiksel araçlar, oyuncuların optimal stratejilerini belirleme ve karar verme süreçlerinde rehberlik etme konusunda önemli bir rol oynar.

İnsan sistemlerindeki çatışma, rekabet, iş birliği ve karşılıklı bağımlılık gibi özellikleri matematiksel olarak modellemek ve analiz etmek için kullanılan bir araçtır. Bu disiplin, stratejik etkileşimleri anlamak, oyuncuların optimal stratejilerini belirlemek ve potansiyel sonuçları tahmin etmek amacıyla kullanılır. Oyun kuramı, birçok alanda uygulanabilir ve karar vericilere stratejik analiz yapma ve en iyi kararları alma konusunda yardımcı olur (Inegbedion ve ark., 2023). Oyun Kuramı, birden fazla oyuncunun farklı seviyelerde yer aldığı karar alma süreçlerinde, bir oyuncunun kararları diğer oyuncuların doğrudan veya dolaylı olarak etkilendiği durumlarda kullanılan en popüler araçlardan biridir. Oyun kuramı, birden fazla oyuncunun stratejik etkileşimini inceleyen ve bir son karar vermek için birçok seçenek olduğu durumlarda kullanılan bir disiplindir (Gupta ve ark., 2023).

Bir oyunda birden fazla oyuncu bulunur ve her bir oyuncu, belirli bir durumda çeşitli eylem seçeneklerine sahiptir. Oyuncular, kendi eylemlerini seçerken diğer oyuncuların eylemlerini de göz önünde bulundururlar, çünkü bu eylemler sonucunda kendi başarıları veya kayıpları etkilenebilir. Oyun kuramı, oyuncuların tercihlerini, hedeflerini ve rasyonel davranışlarını temel alır. Oyuncuların tercihleri, belirli sonuçlara ilişkin değerlendirmelerine dayanır. Örneğin, bir oyuncunun amacı maksimum kazancı elde etmek olabilir veya bir başka oyuncu rekabetçi bir oyunda minimum kayıp yaşamak isteyebilir. Oyun kuramı, oyuncuların bu tercihlerini matematiksel olarak temsil eder. Böylelikle, oyunların stratejik etkileşimlerini analiz edebilme imkânı oluşur. Oyuncular, diğer oyuncuların seçimlerini tahmin ederek ve analiz ederek en iyi eylem

stratejilerini belirlemeye çalışırlar. Oyun kuramı, genellikle oyuncuların çıkarlarının birbirine çeliştiği durumları ele alır. Bu durumda, oyuncular kendi kararlarını verirken, diğer oyuncuların hareketlerini göz önünde bulundururlar. Bu hareketler, oyuncuların kendi başarıları veya kayıpları üzerinde etkili olabilir ve oyuncuların birbirlerine karşı avantaj sağlamak, çatışmaları çözmek veya iş birliği yapmak için stratejik olarak hareket etmelerini sağlar. Oyun ağaçları, oyun matrisleri, Nash denge noktaları, koalisyon teorisi gibi matematiksel model ve araçlar bu amaçla kullanılır. Bu modeller, oyuncuların optimal stratejilerini ve olası sonuçlarını belirlemeye yardımcı olur. Bunlara dayanarak oyun kuramının amacını da açıklayabiliriz.

Oyun kuramının amacı, oyuncuların stratejik etkileşimlerini anlamak ve analiz etmektir. Bu analiz, çeşitli disiplinlerde birçok uygulama alanı bulur. Örneğin, iş dünyasında, rekabet stratejilerini belirlemek ve pazara girmek için en iyi zamanlamayı tahmin etmek için kullanılabilir. Ekonomide, kaynak tahsisinde ve fiyatlandırmada etkili stratejilerin analizine yardımcı olur. Politikada, seçim stratejilerini ve politika etkileşimlerini analiz etmek için kullanılabilir. Bu analizlerle, oyuncuların rasyonel davranışlarını ve stratejik etkileşimlerini anlamak için kullanılan bir çerçeve sağlamış olur. Verdiğimiz örneklere ek olarak, iş dünyasında, rekabet stratejileri belirlemek, pazar payını artırmak ve karlılık elde etmek için kullanılır diyebilirim. Ayrıca, oyun kuramı yapay zekâ ve bilgisayar biliminde de önemli bir rol oynar. Stratejik karar verme, oyun teorisi temelli algoritmaların geliştirilmesi ve oyunlar aracılığıyla yapay zekâ sistemlerinin eğitimi için kullanılır. Oyun kuramı, çatışmaları çözmek, iş birliği yapmak veya rekabet avantajı elde etmek gibi durumlarda stratejik kararlar vermek için bir araç sağlar. Oyuncular, diğer oyuncuların stratejilerini ve potansiyel tepkilerini analiz ederek en iyi eylem stratejilerini belirleyebilirler. Bu, birçok disiplinde karşılaşılan stratejik etkileşimleri anlamak ve analiz etmek için güçlü bir araçtır. Oyun kuramının temel amacı, oyuncuların stratejik etkileşimlerini anlamak, optimal stratejileri belirlemek ve potansiyel sonuçları tahmin etmektir. Bu şekilde, oyuncular, diğer oyuncuların eylemlerine göre en iyi kararları verebilir ve istedikleri hedeflere ulaşabilirler.

Oyun kuramı, insan davranışını anlamak ve açıklamak için de kullanılır. İnsanlar, kararlarını diğer insanların davranışlarını ve motivasyonlarını anlayarak alırlar. Oyun kuramı, bu stratejik etkileşimleri matematiksel bir

çerçeve içinde modelleyerek insan davranışını analiz etmeyi sağlar (Yu, Tseng ve Langari, 2018).

Oyun kuramı birkaç açıdan önemlidir.

Stratejik Karar Alma: Oyun kuramı, stratejik etkileşimlerin olduğu durumlarda oyuncuların optimal kararlarını belirlemelerine yardımcı olur. Bu, iş dünyasında, ekonomide, politikada ve diğer birçok alanda stratejik kararlar verilmesini sağlar (Tengiz, 2020).

Çatışma Çözme: Oyun kuramı, çıkarlarının çeliştiği durumlarda taraflar arasındaki çatışmaların analiz edilmesine yardımcı olur. Oyun kuramı, müzakereler, stratejik iş birliği veya rekabet gibi çeşitli çözüm yaklaşımlarını inceleyerek, çatışmaların sonuçlarını tahmin etmek ve optimize etmek için kullanılır (Prata, 2016; Ayres, 2020).

Pazar Analizi: Oyun kuramı, pazarlardaki rekabeti ve tüketici davranışını anlamak için kullanılır. Oyun kuramı, firmaların fiyatlandırma stratejilerini belirlemesine ve rekabet ortamında kararlarını optimize etmesine yardımcı olur (Hema ve ark., 2023; Bisht ve Dangwal, 2023).

Yapay Zekâ ve Bilgisayar Bilimi: Oyun kuramı, yapay zekâ alanında stratejik karar alma ve oyun teorisi tabanlı algoritmaların geliştirilmesinde önemli bir rol oynar. Bilgisayar bilimi ve yapay zekâ alanındaki birçok algoritma ve yapay zekâ sistemi, oyun kuramının temellerine dayanır (Elkind ve Leyton-Brown, 2010).

Oyun kuramı, stratejik etkileşimleri analiz etmek ve optimal kararlar vermeye yardımcı olmak için güçlü bir araçtır. Oyuncuların farklı stratejileri değerlendirerek, belirli bir durumda en iyi sonuçları elde etmelerini sağlar. Bu, rekabetçi ortamlarda avantaj elde etmek, çatışmaları çözmek veya iş birliği yapmak için kullanılabilir. Ayrıca, oyun kuramı insan davranışını anlamada ve insanlar arasındaki etkileşimleri açıklamada da önemlidir. İnsanlar, kararlarını diğer insanların davranışlarını ve motivasyonlarını anlayarak ve tahmin ederek alırlar. Oyun kuramı, bu tür stratejik etkileşimleri matematiksel bir çerçeve içinde modeller ve analiz eder.

Oyun kuramının önemi aynı zamanda politika analizinde de görülür. Politikacılar, politika ve strateji seçimlerini yaparken diğer aktörlerin tepkilerini ve davranışlarını dikkate almalıdır. Oyun kuramı, politik kararların sonuçlarını ve etkilerini analiz etmek için bir araç olarak kullanılır.

Son 25 yılda, bilgisayar bilimi ve ekonomi arasında yoğun bir etkileşim söz konusu olmuş ve bu etkileşim algoritmik oyun kuramı olarak adlandırılan yeni bir alanın ortaya çıkmasına yol açmıştır. Algoritmik oyun kuramı, geleneksel oyun kuramının bilgisayar bilimi ve algoritmalarla birleştiği bir alan olarak önemli bir rol oynamaktadır (Shoham, 2008). İşte algoritmik oyun kuramının önemli olduğunu gösteren bazı kullanım alanları:

1. ***Bilgisayar Bilimi ve Yapay Zekâ:*** Algoritmik oyun kuramı, bilgisayar bilimi ve yapay zekâ alanında önemli bir uygulama alanıdır. Oyunlar, yapay zekâ algoritmalarının geliştirilmesi, strateji belirleme ve oyun teorisi modellerinin optimize edilmesi için bir ortam sağlar. Algoritmik oyun kuramı, yapay zekanın stratejik kararlar almasını ve oyunlarda rakipleriyle etkileşim halinde olmasını sağlayarak bu alanlarda ilerlemelerin kaydedilmesine katkı sağlar (Elkind ve Leyton-Brown, 2010).

2. ***Dağıtık Sistemler ve Ağlar:*** Algoritmik oyun kuramı, dağıtık sistemler ve ağlar üzerindeki etkileşimleri analiz etmek için kullanılır. Birçok oyuncunun birbirleriyle etkileşim halinde olduğu sistemlerde, algoritmik oyun kuramı, oyuncuların stratejilerini optimize etmeleri ve sistemdeki kaynakların etkin bir şekilde kullanılmasını sağlamaları için kullanılır. Bu, örneğin ağ trafiği yönetimi veya enerji dağıtımı gibi alanlarda önemli bir uygulama alanı bulur (Abraham, Alvisi ve Halpern, 2011; Jiang ve ark., 2022; Chen ve ark., 2023; He ve ark., 2019).

3. ***Optimizasyon ve Verimlilik:*** Algoritmik oyun kuramı, optimizasyon problemlerini çözmede ve kaynakların etkin bir şekilde kullanılmasını sağlamada önemli bir rol oynar. Oyuncuların stratejilerini optimize etmeleri ve en iyi sonuçları elde etmeleri için kullanılan algoritmalar geliştirilir. Bu da kaynakların daha verimli kullanılmasını, maliyetlerin düşürülmesini ve toplam verimin artmasını sağlar (Gunigari ve Chitra, 2023; Neshat ve Amin-Naseri, 2015; Zeng, 2022).

Algoritmik oyun kuramı, çeşitli alanlarda karar verme süreçlerini ve etkileşimleri daha iyi anlamamızı sağlar. Bu, daha iyi stratejilerin belirlenmesine, kaynakların etkin bir şekilde kullanılmasına, rekabetçi ortamlarda avantaj sağlanmasına ve toplumun genel verimliliğinin artırılmasına yardımcı olur. Sonuç olarak, algoritmik oyun kuramı, bilgisayar bilimi, yapay zekâ, pazar analizi, dağıtık sistemler, optimizasyon ve sosyal-ekonomik analiz gibi çeşitli alanlarda önemli roller üstlenmektedir. Bu alanlarda algoritmik oyun

kuramı kullanarak daha iyi stratejilerin belirlenmesi, kaynakların etkin kullanımı, rekabet avantajı elde edilmesi ve toplumun genel verimliliğinin artırılması hedeflenir.

1.2. Algoritmaların, Oyun Kuramındaki Kullanımı

Makine öğrenme algoritmaları ile oyun teorisi arasında ilişki bulunmaktadır. Oyun teorisi, stratejik etkileşimlerin analiz edildiği bir disiplindir ve oyuncuların optimal stratejilerini belirlemek için kullanılır. Makine öğrenme ise bilgisayar sistemlerinin verilerden öğrenme yeteneğine sahip olmasını sağlayan bir alan olarak tanımlanabilir. Makine öğrenme algoritmaları, genellikle veri setlerindeki desenleri tanımak ve bu desenleri kullanarak öngörülerde bulunmak için kullanılır. Oyun teorisi de stratejik etkileşimleri analiz ederken belirli kurallara dayanır ve oyuncuların nasıl davranacaklarına dair tahminlerde bulunur. Günümüzde popüler konulardan olan Blockchain alanında dahi makine öğrenmesi ve algoritmik oyun kuramı ilişkisi kurularak çalışmalar yapılmaktadır (Dey, 2018). Oyun kuramında ve makine öğrenmesinde ki çıkarımlar arasında benzerlikler üzerine eskiden beri çalışmalar yapılmaktadır (Rezek ve ark., 2008). Derin öğrenme ve algoritmik oyun kuramı birlikte kullanılabilir. Böylece Oyun kuramı, çeşitli derin öğrenme temelli problemleri modellemeye veya çözmeye yardımcı olur. (Hazra ve Anjaria, 2022).

Makine öğrenme algoritmaları, oyun teorisindeki stratejik etkileşimleri ve oyuncu davranışlarını modellemek için kullanılabilir. Bir makine öğrenme algoritması, veri setlerinden oyun teorisine dayalı modeller oluşturarak oyuncu davranışlarını tahmin edebilir ve optimal stratejileri belirleyebilir. Örneğin, bir makine öğrenme algoritması, rekabetçi bir oyunun geçmiş performans verilerini analiz ederek oyuncuların stratejilerini tahmin edebilir ve bu bilgiyi kullanarak optimal stratejileri belirleyebilir. Aynı şekilde, bir kooperatif oyun durumunda, makine öğrenme algoritmaları oyuncuların birlikte çalışma eğilimlerini ve stratejik iş birliği yapma olasılıklarını tahmin edebilir.

Bu bağlamda, makine öğrenme algoritmaları, oyun teorisinin analitik araçlarını kullanarak stratejik etkileşimleri ve oyuncu davranışlarını modellemek ve anlamak için kullanılabilir. Makine öğrenme, oyun teorisinin

matematiksel modellerini uygulama ve gerçek dünya senaryolarında optimal kararlar verme konusunda yardımcı olma potansiyeline sahiptir.

Örnek bir İkili rekabetçi oyunda, iki oyuncu (A ve B) sırasıyla 1'den 10'a kadar olan sayılardan birini seçer. Seçilen sayılar toplandığında en büyük toplamı elde etmek için stratejilerini belirlemek zorundadırlar. Her oyuncu, rakibinin seçtiği sayıyı bilmeden kendi seçimini yapar.

Makine öğrenme algoritmaları bu oyunda optimal stratejileri belirlemek için kullanılabilir. Aşağıda, basit bir örnek senaryo yer almaktadır:

1. İlk aşamada, makine öğrenme algoritması rastgele sayılar seçerek bir dizi simülasyon yapar. Örneğin, algoritma A'nın 4, B'nin 7 seçtiği bir simülasyonu gerçekleştirir.
2. Bu simülasyonlardan elde edilen veriler kullanılarak, makine öğrenme algoritması oyuncuların seçimlerini ve toplam sonuçlarını analiz eder.
3. Analiz sonucunda, makine öğrenme algoritması optimal bir strateji belirleyebilir. Örneğin, algoritma, oyuncunun rakibinin son turda seçtiği en büyük sayıyı tahmin ederek ve buna göre en büyük sayıyı seçerek daha yüksek bir toplam elde edebileceğini öğrenebilir.
4. Bu stratejiyi kullanarak, makine öğrenme algoritması bir sonraki simülasyonda daha iyi bir performans gösterebilir ve sürekli olarak optimize edilebilir.

Bu örnek, makine öğrenme algoritmalarının oyun teorisine dayalı stratejileri analiz etmek ve en iyi sonuçları elde etmek için kullanılabileceğini göstermektedir. Oyun teorisi modellerini kullanarak makine öğrenme algoritmaları, stratejik etkileşimleri ve oyuncu davranışlarını anlamada ve optimize etmede yardımcı olabilir. Makine öğrenme algoritmaları, verilerden öğrenme yoluyla modeller oluşturur ve tahminler yapar. Oyun teorisi modelleri, bu algoritmaların geliştirilmesinde ve optimize edilmesinde kullanılabilir. Örneğin, oyun teorisi modelleri, takım halinde öğrenme veya çok oyunculu rekabetçi ortamlarda stratejik kararlar almak için kullanılabilir. Bu modeller, oyuncu davranışlarını anlamaya, oyunun denge noktalarını veya en iyi stratejileri bulmaya ve bu stratejileri optimize etmeye yardımcı olabilir.

Oyun teorisi modelleri, makine öğrenme algoritmalarıyla entegre edildiğinde, özellikle sosyal ve ekonomik sistemlerin analizinde ve yönetiminde önemli bir rol oynayabilir. Bu entegrasyon, daha karmaşık ve gerçekçi problemlerin çözümünde daha etkili sonuçlar elde etmeyi sağlayabilir.

Python dilinde basit bir örnek yapalım. Aşağıdaki örnekte, bir oyunda optimal stratejileri belirlemek için basit bir Python kod örneği verildi.

***Önemli:** Kitapta bulunan bütün Python kodlamalarında, Anaconda.Navigator versiyon: 1.10.0 ve Jupyter Notebook versiyon: 6.1.4 kullanılmıştır.*

Örnek 1:

```
import random
# Oyunu simüle etmek için bir fonksiyon
def simulate_game():
    player_A_choice = random.randint(1, 10)  # Oyuncu A'nın rastgele bir sayı seçmesi
    player_B_choice = random.randint(1, 10)  # Oyuncu B'nin rastgele bir sayı seçmesi
    total = player_A_choice + player_B_choice  # Toplamı hesapla
    return total
# stratejiyi belirleme
def determine_optimal_strategy():
    simulations = 1000  # Simülasyon sayısı
    strategies = {}  # Stratejilerin saklanacağı sözlük
    for _ in range(simulations):
        total = simulate_game()  # Oyunu simüle et
        if total not in strategies:
            strategies[total] = 1
        else:
            strategies[total] += 1
    optimal_strategy = max(strategies, key=strategies.get)  # En çok kazandıran stratejiyi bul
    return optimal_strategy
# Optimal stratejiyi belirleme ve sonucu yazdırma
optimal_strategy = determine_optimal_strategy()
print("En iyi strateji: ", optimal_strategy)
```

```
En iyi strateji:  10
```

Şekil 1. Optimal Stratejileri belirlemek için basit bir Python kodu

Bu örnek kodda, ***simulate_game()*** fonksiyonu rastgele sayılar seçerek oyunu simüle eder ve toplam sonucunu döndürür. ***determine_optimal_strategy()*** fonksiyonu, belirli bir sayıda simülasyon yaparak stratejileri analiz eder ve en iyi stratejiyi belirler. Son olarak, ***optimal_strategy*** değişkeni kullanılarak en iyi strateji ekrana yazdırılır. Yukarıdaki kodda çıktı; En iyi strateji: 10 olarak elde edilmiştir. Çıktı 2 ila 20 arasında bir değer üretecektir.

```python
import numpy as np

# Oyun durumu
game_state = [
    [1, 2, 1],
    [2, 1, 0],
    [0, 0, 0]
]

# Makine öğrenme modeli
model = np.array([
    [0.5, 0.3, 0.2],
    [0.4, 0.1, 0.5],
    [0.6, 0.4, 0.0]
])
```

Şekil 2. 3x3 Matris örnekleri ve değişkenlerin kullanımı

Şekil 2'de yazılan kod parçası, bir oyun durumu ve bir makine öğrenme modelini temsil eden değişkenleri içerir. Kod sadece açıklama için yazılmış olup, çalıştırıldığında herhangi bir çıktı üretmeyecektir.

game_state: *game_state* değişkeni, 3x3'lük bir matrisi temsil eden bir Python listesidir. Bu matris, bir oyunun mevcut durumunu temsil etmektedir. Matristeki değerler oyunun ilgili hücrelerindeki durumu göstermektedir. Örneğin, 1 birinci oyuncuyu, 2 ikinci oyuncuyu ve 0 boş hücreleri temsil eder.

model: model değişkeni, bir makine öğrenme modelini temsil eden bir 3x3 NumPy dizisini içerir. Bu dizi, genellikle bir eğitim süreci sonucunda elde edilen ve bir problemin çözümünü tahmin etmek veya sınıflandırmak için kullanılan parametreleri içerir. Bu parametreler, modelin öğrenilmiş ağırlıklarını veya değerlerini temsil eder. Örnek olarak, sinir ağlarından veya lineer regresyon gibi makine öğrenme algoritmalarından elde edilen katsayıları içerebilir. Örneğin, 3x3'lük model dizisi, bir oyun stratejisini veya bir görüntü sınıflandırma modelini temsil edebilir. Her bir öğe veya hücre, modelin bir durumu veya özelliği temsil eder. Değerler genellikle önceden eğitilmiş bir modelin sonucu olabilir ve daha sonra bu modele dayalı tahminler yapmak veya kararlar vermek için kullanılabilir. Bu örnekteki değerler rastgele olarak atanmıştır ve gerçek bir makine öğrenme modelini temsil etmez. Ancak, gerçek bir makine öğrenme modeli, genellikle eğitim süreci sonucunda optimize edilen ve belirli bir probleme özgü olan değerler içerecektir. Makine öğrenmesi, verilerden öğrenme süreciyle algoritmaların belirli bir görevi veya problemi çözmek için kullanılmasını sağlar. Bu süreç, bir modelin performansını

optimize etmek için yapılan bir optimizasyon sürecini içerir. Bu süreçte, modelin performansını ölçmek için bir hedef fonksiyonu belirlenir ve modelin parametreleri veya ağırlıkları, bu hedef fonksiyonunu maksimize etmek veya minimize etmek için ayarlanır.

Optimizasyon süreci, modelin parametrelerini güncelleyerek eğitim verilerindeki desenleri ve ilişkileri en iyi şekilde yakalamayı hedefler. Bu şekilde, model belirli bir problem veya görevle uyum sağlar ve en iyi sonuçları elde eder. Eğitim sürecinin sonunda, model belirli bir problem veya göreve özgü optimize edilmiş değerlerle temsil edilir.

Bu optimize edilmiş değerler, modelin tahmin yapma yeteneği ve performansını belirler. Gerçek bir makine öğrenme modeli, bu optimize edilmiş değerleri kullanarak yeni girdi verilerine dayalı tahminler yapabilir ve belirli bir problemde en iyi sonuçları elde etmeye çalışır.

Örnek 2'de Python kodunun kullanımını ve kod parçacıklarının işlevini ayrıntılı olarak açıklamaya çalıştım. Temel düzeyde Python bilgisi ile kolayca anlaşılacağını umuyorum.

Örnek 2: Makine öğrenme algoritması ile stratejiyi güncelleme

```
import numpy as np
import random

game_state = np.zeros((3, 3))  # 3x3 boyutunda başlangıç oyun durumu
model = np.zeros((3, 3))  # 3x3 boyutunda başlangıç modeli

def update_strategy():
    """
    Oyun durumuna göre stratejiyi güncelleyen fonksiyon.
    """
    learning_rate = 0.1
    for i in range(len(game_state)):
        for j in range(len(game_state[i])):
            if game_state[i][j] != 0:
                model[i][j] = (1 - learning_rate) * model[i][j] + learning_rate * game_state[i][j]

def choose_move():
    """
    Bilgisayarın hamlesini seçen fonksiyon.
    """
    empty_cells = np.where(game_state == 0)
    indices = list(zip(empty_cells[0], empty_cells[1]))
    row, col = random.choice(indices)
    return row, col

def make_move(row, col, player):
    """
    Belirli bir hücreye hamle yapmayı sağlayan fonksiyon.
    """
    game_state[row][col] = player
```

```
def print_game_state():
    """
    Oyun durumunu ekrana yazdıran fonksiyon.
    """
    for row in game_state:
        print(row)

def check_win(player):
    """
    Oyunun kazananını kontrol eden fonksiyon.
    """
    for i in range(3):
        if game_state[i][0] == game_state[i][1] == game_state[i][2] == player:
            return True
        if game_state[0][i] == game_state[1][i] == game_state[2][i] == player:
            return True
    if game_state[0][0] == game_state[1][1] == game_state[2][2] == player:
        return True
    if game_state[0][2] == game_state[1][1] == game_state[2][0] == player:
        return True
    return False

print("Başlangıç durumu:")
print_game_state()

for _ in range(4):
    row, col = choose_move()
    make_move(row, col, 2)
    print("\nBilgisayarın hamlesi:")
    print_game_state()
    update_strategy()
```

```
    if check_win(2):
        print("\nÜzgünüm, Bilgisayar kazandı.")
        break

    empty_cells = np.where(game_state == 0)
    if empty_cells[0].size == 0:
        print("\nBerabere! Oyun sona erdi.")
        break

    while True:
        row = int(input("\nSatır seçin: "))
        col = int(input("Sütun seçin: "))
        if game_state[row][col] == 0:
            make_move(row, col, 1)
            break
        else:
            print("Geçersiz hamle. Lütfen boş bir hücre seçin.")

    print("\nSizin hamleniz:")
    print_game_state()
    update_strategy()

    if check_win(1):
        print("\nTebrikler! Siz kazandınız.")
        break

    empty_cells = np.where(game_state == 0)
```

Şekil 3. SOS oyunu- basit bir Python kodu

Bu kod, kullanıcının, bilgisayarın ve taşların yer aldığı bir matriste oynamasını sağlar. Oyun 3x3'lük bir matris üzerinde oynanır. Oyuncular sırayla hamle yaparlar ve hedefleri üç taşı yan yana, çapraz veya dikey olarak sıralamaktır. Taşlar S,O ya da 1,2 gibi ifade edilebilir.

- ***game_state:*** Oyunun mevcut durumunu temsil eden 3x3'lük bir matristir. Her hücrede oyuncunun taşı veya boşluk bulunabilir.
- ***model:*** Bilgisayarın stratejisini temsil eden 3x3'lük bir matristir. Bilgisayar, hamle yaparken bu modeli kullanır ve stratejisini günceller.
- ***update_strategy():*** Oyun durumuna göre bilgisayarın stratejisini güncelleyen fonksiyondur. Bilgisayar, oyunun ilerlemesine bağlı olarak stratejisini adapte eder.
- ***choose_move():*** Bilgisayarın bir sonraki hamlesini seçen fonksiyondur. Bilgisayar, stratejisine göre boş bir hücre seçer ve oraya hamle yapar.
- ***make_move(row, col, player):*** Belirli bir hücreye oyuncu tarafından hamle yapmayı sağlayan fonksiyondur. Oyuncu, seçilen hücreye kendi taşını yerleştirir.
- ***print_game_state():*** Oyun durumunu ekrana yazdıran fonksiyondur. Matrisi görüntüler ve mevcut taşları gösterir.
- ***check_win(player):*** Oyunun kazananını kontrol eden fonksiyondur. Matristeki taşları tarar ve bir oyuncunun üç taşı ardışık olarak yerleştirmiş olup olmadığını kontrol eder.

Kod, belirli bir döngü içinde oynanan 3 hamlelik bir oyunu simüle eder. Oyun durumu ve her hamle sonrası strateji güncellemeleri ekrana yazdırılır. Bu şekilde, kodun amacı, makine öğrenme algoritmasının bir oyunda nasıl strateji geliştirebileceğini ve oyun durumuna göre hamleler yapabileceğini göstermektir. Bu tür öğrenme algoritmaları, gerçek oyunlarda strateji geliştirmek veya oyunları oynamak için kullanılabilir. Örnekler arasında satranç, Go, okey gibi oyunlar bulunabilir.

Bu basit Python koduyla yazılan öğrenme algoritmaları ile oyun kuramı arasındaki bağlantıyı kurmak için ufak bir giriş yapmış olduk. Kitapta kullanılacak örnekleri anlayabilmek için makine öğrenme algoritmaları hakkında ve Python yazılım dili hakkında orta seviyede bilgiye ihtiyacınız olacaktır.

İKİNCİ BÖLÜM

2. MATRİS OYUNLARI

Sıfır Toplamlı Model, taraflar arasında bir kaynak veya kazanç paylaşımı söz konusu olduğunda, kazancın bir tarafın kaybıyla doğrudan ilişkili olduğu bir durumu ifade eder. Bu durumda, bir tarafın kazancı diğer tarafın kaybı anlamına gelir ve toplam kazanç veya kayıp sıfır toplamlıdır (Fox, 2010). Örneğin, bir pazarın paylaşımı için rekabet eden iki şirket arasında sıfır toplamlı bir ilişki olabilir. Bir şirketin pazar payını artırması diğer şirketin pazar payını kaybetmesiyle sonuçlanır.

Öte yandan, Sıfır Toplamlı Olmayan Modelde taraflar arasında kazanç veya kayıp paylaşımı farklı şekillerde gerçekleşebilir. Bu modelde, tarafların karlı olabileceği denge durumları ortaya çıkabilir. Tarafların kazancı veya kaybı doğrudan birbirleriyle ilişkili olmayabilir. Örneğin, iş birliği yaparak bir projeyi tamamlayan iki şirket, her iki tarafın da karlı olduğu bir durumu elde edebilir. Bu durumda, tarafların kazançları toplamda sıfır toplamlı olmayabilir, yani her iki taraf da karlı çıkabilir (Fox, 2010).

Sıfır toplamlı olmayan model, iş birliği, stratejik ortaklıklar ve karşılıklı kazanç sağlayan ilişkilerin önemli olduğu durumları ifade eder. Bu tür ilişkilerde taraflar, sadece kendi çıkarlarını değil, aynı zamanda diğer tarafın da çıkarlarını gözeterek hareket ederler.

Matris oyunları, genellikle iki oyunculu sıfır toplamlı oyunlardır ve oyuncuların stratejileri ile getiri matrisinin kesişimlerindeki değerlerle temsil edilirler. Bu oyunlar, matematiksel ve ekonomik analizlerde sıklıkla kullanılan bir modellemedir. Oyuncuların strateji seçimlerine bağlı olarak kararlarının sonuçlarını belirleyen matematiksel bir yapı sunar. Oyuncuların strateji seçimlerini temsil etmek için matrisin satırları kullanılırken, oyuncuların getirileri ise matrisin sütunlarında yer alır. Oyuncuların karşılıklı seçimleri, matrisin kesişim noktalarında bulunan getiri değerleriyle ifade edilir. Pür strateji, bir oyuncunun bir oyun içerisinde belirli bir stratejiyi sabit olarak uygulaması anlamına gelir. Yani oyuncu, her durumda aynı stratejiyi takip eder ve başka bir stratejiye geçiş yapmaz. Pür strateji, bir oyuncunun karar alma sürecindeki kesinlik ve belirsizlik olmadan, her zaman aynı eylemi

gerçekleştireceği anlamına gelir. 1. oyuncunun n tane, II. oyuncunun m tane pür stratejisi olması halinde getiri matrisi aşağıdaki şekilde ifade edilir.

$$A= \begin{array}{c} \\ I_1 \\ I_2 \\ \vdots \\ I_n \end{array} \begin{array}{c} \begin{array}{cccc} II_1 & II_2 & \dots & II_{nm} \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{11} & a_{11} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \end{array}$$

Oyunda 1. oyuncu i. satırı seçer yani I_i stratejisini uygular, buna karşılık II. oyuncu da j. sütunu seçer yani II_j stratejisini uygularsa, i. satır ve j. sütunun kesişimindeki a_{ij} elemanı I. oyuncunun getirisini (kazancını) göstermekte iken oyun sıfır toplamlı olduğundan $-a_{ij}$ değeride II. oyuncunun kaybını göstermektedir. Bu şekilde, matris oyunlarında oyuncuların strateji kombinasyonlarındaki getiri ve kayıpları ifade eden elemanlar, sıfır toplamlı oyunun doğasını yansıtır.

I. oyuncunun pür stratejilerinin kümesi;

$\{I_1, I_2, \dots . I_n\}$ şeklinde ifade edilir.

I. oyuncunun pür stratejilerinin kümesi;

$\{II_1, II_2, \dots . II_m\}$ şeklinde ifade edilir.

Buna göre; (I_i, II_j) ikilisi oyun matrislerinde bir durum belirtmektedir.

Matris oyunları, oyuncuların stratejik kararlarını analiz etmek ve en iyi stratejileri belirlemek için kullanılır. Oyun teorisi, matris oyunlarını analiz etmek için çeşitli yöntemler ve kavramlar sunar, bu da karar verme sürecini anlamak ve oyuncular arasındaki etkileşimleri incelemek için kullanılabilir.

Oyun teorisi, matematiksel ve davranışsal modeller kullanarak oyuncuların karar verme süreçlerini analiz eden bir disiplindir. Oyun teorisi, bir oyunda birden fazla oyuncunun mevcut bilgileri dahilinde stratejilerini belirlemelerini ve bu stratejilere bağlı olarak sonuçların nasıl oluşacağını incelemektedir.

Matris oyunları, oyuncuların bir dizi stratejiye sahip olduğu ve her oyuncunun stratejileri arasından seçim yaparak bir ödül veya kazanç elde etmeye çalıştığı basit bir oyun modelidir. Bu tür oyunlar, oyuncuların hareketlerini ve sonuçlarını bir matris olarak temsil eder.

Bir matris oyununda, oyuncuların stratejileri ve kazançları matrisin satırlarına ve sütunlarına yerleştirilir. Her oyuncu, kendi stratejileri arasından

bir seçim yapar ve diğer oyuncunun seçimine bağlı olarak bir kazanç elde eder. Matrisin her bir hücresi, oyuncuların belirli stratejileri seçtiklerinde elde edecekleri kazançları temsil eder.

Bir matris oyununda I. oyuncunun I_i pür stratejisi ve II. oyuncunun II_j pür stratejisi ile oynadıklarında ortaya çıkan a_{ij} elemanı, aynı anda, matriste bulunduğu satırdaki en küçük ve bulunduğu sütundaki en büyük eleman ise bu taktir de a_{ij} elemanına ***denge noktası*** denir (Genç ve Kadah, 2018).

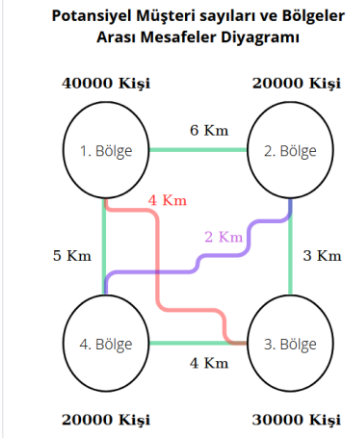
Matris oyunlarını analiz etmek için oyun teorisi çeşitli çözüm kavramları ve yöntemler sunar. Bunlar arasında Nash dengesi, Pareto etkinliği, dominant stratejiler, kooperatif oyunlar ve çok oyunculu oyunlar gibi kavramlar bulunur. Bu yöntemler, oyuncuların en iyi stratejilerini belirlemek, oyun sonuçlarını tahmin etmek ve oyuncular arasında etkileşimleri analiz etmek için kullanılır

Problem: İki mobil oyun firması, dört bölgeden oluşan bir yerde bayi açmak istiyorlar. Aşağıdaki diyagramda bu bölgelerdeki potansiyel müşteri sayıları ve bölgeler arası mesafeler (yollar) gösterilmektedir. X bayisi Y bayisine göre daha fazla tanınan ve güvenilen bir bayidir ve bundan dolayı X bayisinin açtığı bayi Y bayisine göre;

Yakın mesafedeki müşterilerin %70'ini

Eşit uzaklıkta olan müşterilerin %50'sini

Uzakta olan müşterilerin ise %30'unu almakta ve geri kalan müşteriler sigortalarını Y firmasından yaptırmaktadırlar. X ve Y bayilerinin amacı daha fazla sayıda müşteri almak olduğundan getiri matrisini oluşturunuz.



Çözüm: X bayisinin şubesini i bölgesine kurma stratejisini X_i ,

i=1,2,3,4 , ve Y bayisinin şubesini j bölgesine kurma stratejisini Y_j olarak kabul edelim. (X_i, Y_j) durumunda X bayisinin kazanacağı müşteri sayıları cinsinden getiri matrisini beraber oluşturalım.

$(X_1, Y_1) = 40000\,\frac{50}{100} + 20000\,\frac{50}{100} + 30000\,\frac{50}{100} + 20000\,\frac{50}{100} = 55000$

Üstteki eşitlikte $X_1\ ve\ Y_1$ 1. Bölgeye bayi açmış kabul edilmiştir.

$(X_1, Y_2) = 40000\,\frac{70}{100} + 20000\,\frac{30}{100} + 30000\,\frac{30}{100} + 20000\,\frac{30}{100} = 49000$

$(X_2, Y_1) = 40000\,\frac{30}{100} + 20000\,\frac{70}{100} + 30000\,\frac{70}{100} + 20000\,\frac{70}{100} = 61000$

$(X_2, Y_2) = 40000\,\frac{50}{100} + 20000\,\frac{50}{100} + 30000\,\frac{50}{100} + 20000\,\frac{50}{100} = 55000$

$(X_1, Y_3) = 40000\,\frac{70}{100} + 20000\,\frac{30}{100} + 30000\,\frac{30}{100} + 20000\,\frac{30}{100} = 49000$

$(X_3, Y_1) = 40000\,\frac{30}{100} + 20000\,\frac{70}{100} + 30000\,\frac{70}{100} + 20000\,\frac{70}{100} = 61000$

$(X_2, Y_4) = 40000\,\frac{30}{100} + 20000\,\frac{70}{100} + 30000\,\frac{70}{100} + 20000\,\frac{30}{100} = 53000$

$(X_4, Y_2) = 40000\,\frac{70}{100} + 20000\,\frac{30}{100} + 30000\,\frac{30}{100} + 20000\,\frac{70}{100} = 57000$

$(X_1, Y_4) = 40000\,\frac{70}{100} + 20000\,\frac{30}{100} + 30000\,\frac{50}{100} + 20000\,\frac{30}{100} = 55000$

$(X_4, Y_1) = 40000\,\frac{30}{100} + 20000\,\frac{70}{100} + 30000\,\frac{50}{100} + 20000\,\frac{70}{100} = 55000$

$(X_2, Y_3) = 40000\,\frac{30}{100} + 20000\,\frac{70}{100} + 30000\,\frac{30}{100} + 20000\,\frac{70}{100} = 49000$

$(X_3, Y_2) = 40000\,\frac{70}{100} + 20000\,\frac{30}{100} + 30000\,\frac{70}{100} + 20000\,\frac{30}{100} = 61000$

$(X_3, Y_3) = 40000\,\frac{50}{100} + 20000\,\frac{50}{100} + 30000\,\frac{50}{100} + 20000\,\frac{50}{100} = 55000$

$(X_3, Y_4) = 40000\,\frac{70}{100} + 20000\,\frac{30}{100} + 30000\,\frac{70}{100} + 20000\,\frac{30}{100} = 61000$

$(X_4, Y_3) = 40000\,\frac{30}{100} + 20000\,\frac{70}{100} + 30000\,\frac{30}{100} + 20000\,\frac{70}{100} = 49000$

$(X_4, Y_4) = 40000\,\frac{50}{100} + 20000\,\frac{50}{100} + 30000\,\frac{50}{100} + 20000\,\frac{50}{100} = 55000$

Getiri Matrisi =

$$\begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{array} \begin{array}{c} \begin{array}{cccc} Y_1 & Y_2 & Y_3 & Y_4 \end{array} \\ \begin{bmatrix} 55000 & 49000 & 49000 & 55000 \\ 61000 & 55000 & 49000 & 53000 \\ 61000 & 61000 & 55000 & 61000 \\ 55000 & 57000 & 49000 & 55000 \end{bmatrix} \end{array}$$

Matris oyunları, oyun teorisi alanında önemli bir araç olarak kabul edilir ve stratejik karar alma durumlarını analiz etmek için yaygın bir modelleme

aracıdır (Guseinov, Akyar ve Düzce, 2010). Matris oyunlarının temel kavramlarını açıklamak isterim.

2.1. Oyuncular

Bir matris oyununda en az iki oyuncu bulunur. Her oyuncu, kendi stratejilerini belirler ve bu stratejiler arasından seçim yapar. Oyuncuların hedefi, kendi seçtikleri stratejilere bağlı olarak maksimum kazancı elde etmektir. Oyuncular, matris oyunlarında yer alan bireyler veya gruplar olarak düşünülebilir. Her bir oyuncu, oyunda belirli bir hedefi olan ve bu hedefe ulaşmak için stratejiler geliştiren bir aktördür (Köse, 2014).

Örneğin, bir ticaret oyununu düşünelim. Bu oyunda iki oyuncu, A ve B, birbirleriyle rekabet halindedir. Oyuncu A, bir malı yüksek fiyatla satmak istemektedir, oyuncu B ise aynı malı düşük fiyatla satın almak istemektedir. Her iki oyuncu da kendi çıkarlarını en üst düzeye çıkarmak için stratejiler geliştirecektir.

Oyuncuların stratejileri, karar verme sürecinde aldıkları aksiyonları veya hamleleri temsil eder. Bu stratejiler, belirli bir durumda ne yapacaklarını, hangi seçeneği seçeceklerini veya hangi aksiyonu gerçekleştireceklerini gösterir. Matris oyunlarında, her oyuncunun strateji seti ve bu stratejiler arasında seçim yapma yeteneği vardır. Oyuncular, stratejilerini rakiplerinin stratejilerine karşı optimize etmek ve kendi kazançlarını en üst düzeye çıkarmak için çeşitli kombinasyonlarını deneyebilirler. Matris oyunlarında temel hedef, her oyuncunun maksimum kazancı elde etmek için en iyi stratejilerini belirlemektir. Bu stratejileri belirlemek için oyuncular, oyun teorisindeki çeşitli analiz yöntemlerini kullanabilirler ve Nash dengesi gibi denge noktalarını arayabilirler.

2.2. Stratejiler

Stratejiler, oyuncunun belirli bir hamle veya aksiyonu temsil eder. Örneğin, bir oyuncu "A" veya "B" gibi iki stratejiye sahip olabilir. Oyuncu, bu stratejiler arasından birini seçer ve uygular. Oyuncuların belirli bir durumda ne yapacaklarına ilişkin kararlarını ifade eder. Her oyuncunun farklı strateji seçenekleri vardır ve bu seçenekler oyuncuların hedeflerini gerçekleştirmek için benimsedikleri aksiyonları temsil eder.

Bir matris oyununda stratejiler, genellikle oyuncuların alabileceği farklı hamleler veya seçenekler olarak düşünülebilir. Bu hamleler, oyuncuların diğer oyuncuların hamlelerine bağlı olarak kazançlarını veya kayıplarını etkileyebilir (Rençber, 2012).

Örneğin, basit bir savaş stratejisi oyununu ele alalım. Oyunda iki oyuncu, A ve B, belirli bir durumda birliklerini saldırmaya veya savunmaya yönlendirebilirler. Her oyuncunun iki farklı stratejisi vardır: "Saldırı" ve "Savunma".

Oyuncu A, diğer oyuncunun saldırdığı durumlarda "Saldırı" stratejisini benimseyebilirken, diğer oyuncunun savunduğu durumlarda "Savunma" stratejisini benimseyebilir. Benzer şekilde, oyuncu B de aynı stratejileri uygulayabilir.

Matris oyunlarında, oyuncuların stratejileri ve bu stratejilerin sonuçları genellikle bir kazanç matrisi veya ödül matrisi şeklinde temsil edilir. Kazanç matrisi, her oyuncunun her bir stratejik kombinasyon için elde edebileceği kazançları gösterir. Bu matris, oyuncuların strateji seçimlerini optimize etmelerine yardımcı olur.

2.3. Kazançlar

Kazançlar, oyuncuların her bir strateji kombinasyonu için elde edebilecekleri sonuçları veya ödülleri ifade eder. Matris oyunlarında, her oyuncunun her bir stratejisi için bir kazanç değeri belirlenir. Bu değerler, oyuncuların hedeflerine bağlı olarak farklılık gösterebilir ve genellikle bir kazanç matrisi veya ödül matrisi şeklinde temsil edilir (Çiftçi, 2017).

Örneğin, bir rekabetçi futbol maçını düşünelim. Oyuncu A ve oyuncu B, farklı stratejiler uygulayarak kazanç elde etmeye çalışırlar. Her bir oyuncunun kazanç matrisi, farklı stratejik kombinasyonlarda elde edebilecekleri sonuçları gösterir.

2.4. Kazanç Matrisi

Kazanç matrisi, oyuncuların stratejilerini ve bu stratejiler arasındaki etkileşimlerden kaynaklanan kazançları içerir. Genellikle satırlar bir oyuncunun stratejilerini temsil ederken, sütunlar diğer oyuncunun stratejilerini temsil eder. Her hücredeki değer, o stratejileri seçtiklerinde oyuncuların elde

edecekleri kazancı gösterir. Kazanç matrisi, oyuncuların her bir strateji kombinasyonu için elde edebilecekleri kazanç veya kayıpları gösteren bir matristir.

Kazanç matrisi, matris oyunlarında oyuncuların kararlarını ve stratejilerini belirlemek için temel bir araçtır. Her bir oyuncunun her bir stratejisi için birer satır ve sütun olmak üzere bir matris şeklinde temsil edilir. Oyuncu sayısına bağlı olarak, kazanç matrisi iki boyutlu olabilir (2 oyunculu oyunlar için) veya daha fazla boyutta olabilir (Clemente, Fernández, Puerto, 2011).

Bir kazanç matrisi örneğiyle açıklamak gerekirse, bir ikili oyunculu oyunu ele alalım. Oyuncu A ve oyuncu B, farklı stratejiler uygulayarak kazanç elde etmeye çalışırlar. Her bir oyuncunun stratejileri, satır ve sütun başlıkları altında temsil edilir ve matrisin hücrelerinde kazanç değerleri yer alır.

2.5. Oyunlar

Oyunlar, oyuncuların birbirleriyle etkileşim halinde olduğu ve kararlar alarak sonuçlar elde ettiği modellerdir. Oyunlar, oyuncuların stratejilerini optimize etme amacıyla birbirleriyle rekabet ettiği veya iş birliği yaptığı durumları temsil eder. Şimdi stratejilerden ve beklenen değerden bahsetmek isterim.

2.5.1. Tam Arı Stratejiler (Pür Stratejiler)

Bir oyuncunun her durumda sadece bir stratejiyi seçmesi durumunda, oyun tam arı stratejilere sahiptir. Bu durumda, oyuncuların kararları tam olarak belirlenir ve oyunun sonucu tamamen stratejilere bağlıdır. Örneğin, bir oyunda her oyuncunun sadece "Kazanma" veya "Kaybetme" stratejilerinden birini seçmesi tam arı stratejilere örnektir (Polat, 2021).

2.5.2. Karma Stratejiler

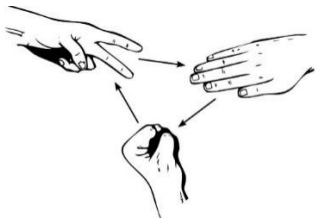
Bir oyuncunun duruma bağlı olarak farklı stratejileri seçmesine izin verildiğinde, oyun karma stratejilere sahiptir. Oyuncular, belirli bir olasılık dağılımıyla farklı stratejileri benimseyebilir. Karma stratejiler, belirli bir oyuncunun diğer oyuncunun stratejilerini tahmin edemediği durumlarda

kullanılır. Örneğin, bir oyunda bir oyuncunun "Kazanma" veya "Kaybetme" stratejilerini belirli bir olasılıkla seçmesi karma stratejilere örnektir. Oyuncuların karma stratejileri kullanarak riski dağıtması ve daha dengeli sonuçlar elde etmesi mümkün olabilir. Oyun teorisi, karma stratejileri analiz etmek ve oyuncu davranışlarını anlamak için önemli bir araç sağlar (Acar ve Ünal, 2022).

2.5.3. Beklenen Değer

Oyuncular, stratejilerine bağlı olarak elde edecekleri beklenen değeri hesaplayabilirler (Uysal, Gülmez ve Çucukçu, 2017). Beklenen değer, her bir strateji kombinasyonunda oyuncunun elde edebileceği kazancın veya kaybın bir ölçüsüdür. Oyuncular, en yüksek beklenen değeri elde edecek stratejileri seçmeye çalışırlar.

Hemen bir örnek vererek Python kodunu paylaşayım. İki oyunculu bir "Taş-Kâğıt-Makas" oyununu ele alalım. Bu oyunu oynayan iki oyuncu (Player I ve Player II) arasında etkileşim vardır. Her oyuncu, "Taş", "Kâğıt" veya "Makas" gibi üç farklı stratejiyi seçebilir.



Player I stratejileri: {Taş (Rock), Kâğıt (Paper), Makas (Scissors)}
Player II stratejileri: {Taş (Rock), Kâğıt (Paper), Makas (Scissors)}
Kazanç matrisi aşağıdaki gibi olabilir:

		Player II		
		A	B	C
Player I	A	(a_{11}, b_{11})	(a_{12}, b_{12})	(a_{13}, b_{13})
	B	(a_{21}, b_{21})	(a_{22}, b_{22})	(a_{23}, b_{23})
	C	(a_{31}, b_{31})	(a_{32}, b_{32})	(a_{33}, b_{33})

Şekil 4. 2 oyunculu 3 hamleli oyunun kazanç matrisi

		Player II		
		rock	paper	scissors
Player I	rock	(0, 0)	(-1, 1)	(1, -1)
	paper	(1, -1)	(0,0)	(-1,-1)
	scissors	(-1, 1)	(1, -1)	(0, 0)

Şekil 5. Taş, Kâğıt, Makas oyununun kazanç matrisi

Bu kazanç matrisinde, her bir strateji kombinasyonu için oyuncuların kazanç veya kayıplarını temsil eden sayılar bulunur. Örneğin, Player I Taş seçerse ve Player II Kâğıt seçerse, Player I -1 birim kazanırken Player II 1 birim kazanır. -1 kaybetme, 0 berabere, 1 kazanma anlamı taşımaktadır.

Bu örnekte, Player I ve Player II oyuncuları için hem tam arı stratejiler hem de karma stratejiler uygulanabilir. Eğer oyuncular tam arı stratejileri kullanıyorlarsa, her bir oyuncu her durumda sadece bir stratejiyi seçer. Ancak, karma stratejiler kullanıldığında, oyuncular farklı stratejileri belirli bir

olasılıkla seçebilir. Karma stratejileri kullanarak, oyuncular beklenen değeri hesaplayabilirler. Beklenen değer, her bir strateji kombinasyonunda oyuncunun elde edeceği kazancın veya kaybın bir ölçüsüdür. Oyuncular, en yüksek beklenen değeri elde edecek stratejileri seçmeye çalışırlar. Karma stratejilerden ileride bahsedilecektir. Aşağıda bir örnek Python kodu veriyorum. Bu kod, "Taş-Kâğıt-Makas" oyunu onamaktadır (İzgi ve Özkaya, 2019). Oyun, kazanç matrisini kullanarak bir makine öğrenme algoritması olan "Q-learning" algoritmasını uygulamaktadır. Q-learning algoritması, bir takımın deneyimlerinden öğrenerek, karar verme sürecinde en iyi aksiyonları seçmeyi öğrenen bir takım tabanlı güçlendirme öğrenme algoritmasıdır. "Takım tabanlı güçlendirme öğrenme algoritması" terimi, bir grup oyuncunun birlikte çalışarak belirli bir hedefi elde etmeyi öğrenmek için güçlendirme öğrenme yöntemlerini kullanabileceği bir algoritmayı ifade eder. Ama kullanılmayan örnekleri de mevcuttur. Bu tür bir algoritma, takımdaki her bir oyuncunun kendi eylemlerini gerçekleştirerek ortak hedefe doğru ilerlemesini sağlar ve ödüllendirme ve cezalandırma mekanizmalarını kullanarak performanslarını optimize etmeyi hedefler. Bu sayede, takımın bütün olarak daha iyi sonuçlar elde etmesi ve iş birliği içinde çalışması amaçlanır. Ama her zaman böyle olmayabilir. Çoklu oyunlarda daha çok uygulanabilen bu durum, iki kişinin de kendi kazancını gözeterek oynadığı oyunlarda farklılık gösterebilir.

Örnek 3: "Taş-Kâğıt-Makas" oyunu için Python kodu (Q-Learning) yazalım.

```python
import random

# Kazanç matrisi
payoff_matrix = [[0, -1, 1], [1, 0, -1], [-1, 1, 0]]

# Q-learning algoritması
Q = [[random.random() * 0.1 for _ in range(3)] for _ in range(3)]
learning_rate = 0.1  # Öğrenme hızı
discount_factor = 0.9  # İndirim faktörü
num_episodes = 1000  # Toplam bölüm sayısı

# Q-learning algoritmasının uygulanması
for episode in range(num_episodes):
    state = random.randint(0, 2)  # Rastgele bir başlangıç durumu seç
    while True:
        action = max(range(3), key=lambda x: Q[state][x])  # En yüksek Q değerine sahip aksiyonu seç
        next_state = random.choice([0, 1, 2])  # Rastgele bir sonraki durum seç
        reward = payoff_matrix[state][action]  # Ödül hesapla
        Q[state][action] += learning_rate * (reward + discount_factor *
                                             max(Q[next_state]) - Q[state][action])  # Q değerini güncelle
        state = next_state
        if state == 0:  # Oyunu tamamladığımızda döngüyü sonlandır
            break

# Eğitim sonucunda elde edilen Q değerleri
print("Q Değerleri:")
for row in Q:
    print(row)
```

```
Q Değerleri:
[0.03406477349219409, 0.04728567634915423, 9.987205730377001]
[9.989391817727363, 0.0034119267359769556, 0.010875117854806372]
[-0.02310429636408487, 9.988702349623313, 0.024178761116781668]
```

Şekil 6. "Taş-Kâğıt-Makas" oyunu için Python kodu ve Çıktısı (Q-Learning)

Önce Q-learning algoritmasının kullanım şekilleri ve çalışma mantığı üzerine açıklama yapıp sonra kodun ne yaptığını ayrıntılı olarak açıklayacağım. Bu kod, bir Q-learning algoritması kullanarak verilen bir kazanç matrisi üzerinde eğitim yapmayı amaçlar ve oyuncuların stratejilerini optimize etmeye çalışır. Q-learning, bir durum-aksiyon tablosu (Q tablosu) kullanarak, her durum için aksiyonların değerini tahmin eder. Başlangıçta Q değerleri sıfır olarak başlatılır ve algoritma deneme yanılma yoluyla Q değerlerini günceller. Q-learning, yaygın olarak kullanılan bir pekiştirmeli öğrenme algoritmasıdır ve yapay zekâ programlarının (açıklayıcı bir şekilde "agent" olarak adlandırılır) gelecek planlamasını yapmasına olanak sağlamak için sıkça kullanılır. Bu, teşvik prensibine dayanır ve günlük hayatta sıkça kullandığımız teşvik prensibiyle benzerlik gösterir. Teşvikler bir kişiyi veya bir kurumu belirli bir göreve yönlendirebiliyorsa, Q-learning de aynı şekilde bir robotu veya başka bir yapay zekâ varlığını belirli bir hedefe yönlendirebilir.

Algoritma, temel olarak iki matris yapısına dayanır: ödül matrisi (R) ve Q değeri matrisi (Q). Belirli bir iterasyon süresi boyunca robot, çevre içinde eylemler gerçekleştirir ve Q matrisini, ödül matrisindeki değerlere dayanarak

günceller. Tüm iterasyonların sonunda, Q matrisindeki nihai değerler bize en optimal sonucu sağlar (Zhang ve ark., 2019).

Her bir bölümde (episode), oyuncular rastgele bir başlangıç durumu seçer ve oyunu oynamaya başlar. Her adımda, oyuncu mevcut durumu dikkate alarak en yüksek Q değerine sahip aksiyonu seçer ve rastgele bir sonraki durumu belirler. Kazanç matrisine göre elde edilen ödülü hesaplar ve Q değerini günceller. Bu işlem, oyun tamamlanana kadar devam eder. Sonuç olarak, Q-learning algoritması, eğitim bölümleri boyunca Q değerlerini güncelleyerek en iyi stratejileri öğrenir. Sonunda, Q tablosunda elde edilen değerler, oyuncuların en iyi stratejilerini temsil eder.

Bu örnek, Q-learning algoritmasının "Taş-Kâğıt-Makas" oyununda nasıl çalışabileceğini göstermektedir. Oyuncuların stratejilerini optimize etmek için Q-learning algoritması kullanılır (Shi ve Rong, 2022).

Kodun çalışma mantığı şu şekildedir:

1. İlk olarak, kazanç matrisi payoff_Matris oluşturulur. Bu matris, her iki oyuncunun farklı stratejilerle birbirine karşı elde edeceği kazançları içerir.
2. Q-learning algoritması için başlangıç Q tablosu Q oluşturulur. Başlangıçta tüm değerler sıfır olarak ayarlanır.
3. Ardından, belirli sayıda bölüm (episode) için aşağıdaki adımlar tekrarlanır:
    - Rastgele bir başlangıç durumu seçilir.
    - Oyuncular, mevcut duruma göre en yüksek Q değerine sahip aksiyonu seçer.
    - Rastgele bir sonraki durum seçilir.
    - Elde edilen ödül, payoff_Matris'ten alınır.
    - Q değeri güncellenir: Yeni Q değeri, önceki Q değeri, ödül, indirim faktörü ve sonraki durumdaki en yüksek Q değeri kullanılarak hesaplanır.
4. Eğitim bölümleri tamamlandıktan sonra, Q tablosu ekrana yazdırılır. Bu tablo, her bir durum ve aksiyon kombinasyonu için elde edilen Q değerlerini içerir.

Bu şekilde, Q-learning algoritması oyunları öğrenerek en iyi stratejileri belirlemeye çalışır. Örnek kodda kullanılan "Taş-Kâğıt-Makas" oyunu için

kazanç matrisi ve Q-learning algoritması, oyuncuların en iyi stratejilerini bulmalarına yardımcı olur.

Çıktıdaki Q Değerleri matrisi, Q-learning algoritmasının eğitimi sonucunda elde edilen Q değerlerini temsil eder. Her satır, bir durumu (Taş, Kâğıt, Makas) temsil ederken, her sütun bir aksiyonu (Taş, Kâğıt, Makas) temsil eder. Örneğin, Şekil 6'daki kod çalıştırıldıktan sonra çıktıda gördüğümüz [0.03406477349219409, 0.04728567634915423, 9.987205730377001] satırı Player I Taş durumunda iken her aksiyon (Taş, Kâğıt, Makas) için Q değerlerini temsil eder. Benzer şekilde, diğer iki satır da Player I'in Kâğıt ve Makas durumları için Q değerlerini temsil eder. Q değerleri, her durum-aksiyon çifti için o durumda o aksiyonun seçilmesinin beklenen faydasını temsil eder. Daha yüksek Q değerleri, o durumda o aksiyonun daha olumlu sonuçlar getirdiğini gösterir. Eğitim süreci boyunca, Q değerleri güncellenir ve stratejiyi optimize etmek için kullanılır.

"Taş-Kâğıt-Makas" oyunu için Q-Learning algoritmasını kullanmadan direk sonuca giderek çıktıyı kazanan olarak görebileceğimiz Python kodunu yazalım.

Örnek 4: "Taş-Kâğıt-Makas" oyunu için Python kodu yazalım.

```
win_matrix = {
    "taş": {"taş": 0, "kağıt": -1, "makas": 1},
    "kağıt": {"taş": 1, "kağıt": 0, "makas": -1},
    "makas": {"taş": -1, "kağıt": 1, "makas": 0}
}

oyuncu_A = input("Oyuncu A, taş, kağıt veya makas seçin: ").lower()
oyuncu_B = input("Oyuncu B, taş, kağıt veya makas seçin: ").lower()

if oyuncu_A not in win_matrix or oyuncu_B not in win_matrix:
    print("Geçersiz hamle. Sadece taş, kağıt veya makas seçebilirsiniz.")
else:
    sonuç = win_matrix[oyuncu_A][oyuncu_B]
    if sonuç == 0:
        print("Berabere!")
    elif sonuç == 1:
        print("Oyuncu A kazandı!")
    else:
        print("Oyuncu B kazandı!")
```

```
Oyuncu A, taş, kağıt veya makas seçin: Taş
Oyuncu B, taş, kağıt veya makas seçin: Makas
Oyuncu A kazandı!
```

Şekil 6. "Taş-Kâğıt-Makas" oyunu için Python kodu ve Çıktısı

Kodumuzda Numpy kütüphanesine veya herhangi bir dış pakete ihtiyaç duyulmamaktadır. Numpy kullanmadan bu tür basit karşılaştırmaları sizlerde gerçekleştirebilirsiniz. ***win_Matris***, oyuncuların hamlelerinin kazançlarını temsil eden bir sözlük yapısıdır. Her oyuncunun her hamle için kazanç değeri belirtilmiştir.

Bu kodda, kullanıcıdan iki oyuncunun hamlelerini girmesi istenir. Ardından, girilen hamleler win_Matris sözlüğü kullanılarak karşılaştırılır ve sonuç ekrana yazdırılır. Eğer girilen hamleler geçersiz ise (taş, kâğıt veya makas olmayan bir değer) hata mesajı verilir.

ÜÇÜNCÜ BÖLÜM

3. ALGORİTMİK OYUN KURAMININ TEMEL MANTIĞI VE OYUN ÇEŞİTLERİ

Oyunların sonuçlarını belirlemek için, ödemeler matrisi üzerinde çözüm süreci gerçekleştirilir. Bu süreç, hangi oyuncunun perspektifinden oyunun değerlendirileceğinin seçimiyle başlar. Eğer çözüm satırları temsil eden oyuncu için yapılacaksa, "maximin" yöntemi kullanılır. Sütunları temsil eden oyuncu için çözüm yapılacaksa ise "minimax" yöntemi uygulanır. Eğer maximin ve minimax değerleri birbirine eşitse, oyun "arı stratejili" olarak kabul edilir.

Maximin yönteminde, öncelikle ödemeler matrisinin her satırının en küçük elemanı seçilir. Ardından, bu seçilen en küçük değerler arasından en büyüğü belirlenir. Bu bulunan değer, ödemeler matrisindeki satırları temsil eden oyuncunun beklenen değeri olarak kabul edilir. Çünkü bu oyuncu, diğer oyuncunun en büyük değeri seçmeyeceğini ve diğer oyuncunun oyundan çekileceğini bilir. Dolayısıyla, bu oyuncu açısından en küçük değerlerin en büyüğü mantıklı bir sonuç olarak kabul edilir. Başka bir deyişle, bu oyuncu açısından mevcut strateji, "kötülerin en iyisi" olarak özetlenebilir.

Sütunları temsil eden oyuncunun perspektifinden bakıldığında ise doğru mantık, "iyilerin kötüsü" olacaktır. Çünkü sütunları temsil eden oyuncu, diğer oyuncunun maximin stratejisini bildiği için oyunu minimax stratejisiyle oynar. Bu oyuncu, elemanları gözden geçirir ve her sütunun en büyük değerini seçer. Bu oyuncu açısından, oyunun sonucu bu seçilen değerlerin en küçüğüdür.

3.1. Minimax ve Maximin Teoremleri

Minimax ve maximin teoremleri, oyun kuramında strateji seçiminde kullanılan önemli kavramlardır. İkisi de oyunlardaki stratejik kararların temelini oluşturur (Fang ve ark., 2021). I. Oyuncu $\{I_1, I_2, I_3, I_4 \dots\dots I_{n-1}, I_n\}$ pür stratejilerini ve II. Oyuncu $\{II_1, II_2, II_3, II_4 \dots\dots II_{m-1}, I_m\}$ pür stratejilerini kullanarak bir amaç belirlerler;

I. oyuncu kazançlarını arttırmayı amaçlarken II. oyuncu ise, I. oyuncunun kazançlarını azaltmaya yani kendi kaybını minimal düzeyde tutmayı amaçlamaktadır.

Başka bir değişle oyun sıfır toplamlı olduğundan, I. oyuncu a_{ij}'leri maksimumda tutmayı, II. oyuncu $-a_{ij}$'leri minimumda tutmayı amaçlamaktadır. Yani II. oyuncu I. oyuncunun kazancını minimalleştirmeyi amaçlamaktadır.

$$\begin{matrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{11} & a_{11} & \dots & a_{2m} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{matrix} \quad \gg\gg\gg \quad \begin{matrix} min\ a_{1j} \\ min\ a_{2j} \\ \vdots \\ min\ a_{nj} \end{matrix}$$

$$\lll$$

$$max\ a_{i1} \quad max\ a_{i2} \quad \dots \quad max\ a_{im}$$

Yukarıdaki matriste max min a_{ij}birinci oyuncunun kazançlarının alt sınırını göstermektedir. min max a_{ij} II. oyuncunun kayıplarının üst sınırını göstermektedir.

3.1.1. Minimax Teoremi:

Minimax teoremi, bir oyunda iki oyuncu arasındaki rekabetçi bir durumu ifade eder. Bu teorem, oyuncuların "en kötü durumu" en aza indirmek için en iyiyi seçeceğini varsayar (Kjeldsen, 2001).

Minimax teoremi, sütunları temsil eden oyuncunun oyunu maksimize etmek ve satırları temsil eden oyuncunun oyunu minimize etmek istediği durumları içerir. Sütun oyuncusu, kendi stratejilerini belirlerken, satır oyuncusunun en iyi cevabını tahmin eder ve kendi kazancını maksimize etmeye çalışır. Satır oyuncusu ise, sütun oyuncusunun en kötü tahminini yapar ve kendi kaybını minimize etmeye çalışır. Bu şekilde, her iki oyuncu da en iyi sonuca ulaşmak için karşı oyuncunun stratejilerini dikkate alır. II. oyuncunun j. Stratejiyi seçtiğini kabul edelim. Bu demektir ki II. oyuncu j. sütundaki elemanlardan herhangi birisini I. oyuncuya vermeyi kabul etmiş demektir. I. oyuncu kazancını maksimum düzeyde tutmayı amaçlayacaktır. Bu da demektir ki I. oyuncu max a_{ij} kadar bir kazancı elde etmiş olur. Yani II. oyuncunu j. pür stratejisi uyguladığında amaçladığı kayıp max a_{ij} kadardır. II.

oyuncu kaybını minimal düzeyde tutmak için min max a_{ij} stratejisini seçecektir.

3.1.2. Maximin Teoremi:

Maximin teoremi, bir oyunda bir oyuncunun kendisi için en iyi durumu garanti etmek amacıyla en kötü durumu en aza indireceğini ifade eder. Bu teorem, oyuncunun risk almaktan kaçınarak garantili bir minimum kazanç elde etmeyi hedeflediği durumları kapsar (Holler, 1990).

Maximin teoremi, satırları temsil eden oyuncunun oyunu minimize etmek ve sütunları temsil eden oyuncunun oyunu maksimize etmek istediği durumları içerir. Satır oyuncusu, sütun oyuncusunun en iyi tahminini yapar ve kendisi için en kötü durumu minimize etmeye çalışır. Sütun oyuncusu ise, satır oyuncusunun en kötü tahminini tahmin eder ve kendi kazancını maksimize etmeye çalışır. Bu şekilde, her iki oyuncu da riski en aza indirgemek için karşı oyuncunun stratejilerini dikkate alır.

I. oyuncunun mevcut pür stratejilerinden i. stratejiyi seçmiş olduğunu kabul edelim. Bu demektir ki I. oyuncu i. satırda denk gelen elemanı amaçlamaktadır. Buna karşılık II. oyuncu I. oyuncunun kazançlarını minimum düzeyde tutmayı amaçladığı için i. satırdaki min a_{ij} biçimindeki j stratejisini seçmeyi tercih eder. Burada min dediğimiz j = {1,2,3,4, ……, m-1, m} değerlerini temsil etmektedir. Dolayısıyla, I. oyuncu min a_{ij} kadar kazancı garantilemiş olmaktadır. Biliyoruz ki I. oyuncu kazancını maksimum düzeyde tutmayı amaçlamaktadır. Kendisi için en iyi kazancı veren strateji max min a_{ij} stratejisi olacaktır. Burada max dediğimiz i = {1,2,3,4, ……, n-1, n} değerlerini temsil etmektedir. Dolayısıyla I. oyuncunun kazancı kesinlikle max min a_{ij} kazancından az olamaz.

Minimax ve maximin teoremleri, oyunlarda strateji seçiminde kullanılan temel prensipleri temsil eder. Bu teoremler, oyuncuların kararlarını verirken diğer oyuncunun stratejilerini hesaba katmalarını sağlar ve Nash dengesi gibi oyunun denge noktalarını belirlemede önemli bir rol oynar (Tanaka, 1994).

Gönül Su ve Rüzgar adında iki iş insanı düşünelim. İkisi de bir proje üzerinde çalışıyorlar ve projenin sonucu üzerinde rekabet halindeler. Projeyi tamamlamak veya başarısız olmak için iki seçenekleri var.

Kazanç matrisi aşağıdaki gibi olsun:

	Gönül Su	Rüzgar
Tamamla	5	-3
Başarısız	-1	2

Bu matriste, Gönül Su'nun kazançları ilk sütunda, Rüzgar'ın kazançları ise ikinci sütunda temsil edilmektedir. Gönül Su ve Rüzgar'ın seçim yapması gereken iki strateji vardır: "Tamamla" veya "Başarısız".

Minimax'a göre: Rüzgar, minimax stratejisini kullanarak karar verecektir. Onun amacı, en kötü durumu en aza indirmektir. Rüzgar, kendi kazançlarını maksimize etmeye çalışırken Gönül Su'nun en kötü tahminini yapar. Bu durumda, Rüzgar'ın en iyi seçeneği "Başarısız" stratejisidir, çünkü Gönül Su'nun "Başarısız" seçeneğinde daha düşük bir kazancı vardır. Rüzgar, bu şekilde en kötü durumu en aza indirir.

Maximin'e göre: Gönül Su, maximin stratejisini kullanarak karar verecektir. Onun amacı, kendisi için en iyi durumu garanti etmektir. Gönül Su, riski en aza indirgemek ve garantili bir minimum kazanç elde etmek istediği için maximin stratejisini benimser. Gönül Su, sütun oyuncusu olarak Rüzgar'ın en iyi tahminini yapar ve kendisi için en kötü durumu minimize etmeye çalışır. Bu durumda, Gönül Su'nun en iyi seçeneği "Tamamla" stratejisidir, çünkü Rüzgar'ın "Tamamla" seçeneğinde daha düşük bir kazancı vardır. Gönül Su, bu şekilde garantili bir minimum kazanç elde eder.

Sonuç olarak, Rüzgar minimax stratejisini kullanarak "Başarısız" seçeneğini tercih ederken, Gönül Su maximin stratejisini kullanarak "Tamamla" seçeneğini tercih eder. Minimax ve maximin stratejileri, oyuncuların risk toleransı ve rakibin stratejileri hakkındaki tahminlere dayalı olarak karar vermesine olanak sağlar.

Başka bir örnek daha vereyim. Ali ve Veli bir iş birliği yapmak istiyorlar ve iki seçenekleri var: "A" veya "B" seçeneklerini seçmek. Kazanç matrisi aşağıdaki gibi olsun:

	Ali	Veli
A	2	5
B	4	3

Bu matriste, Ali'nin kazançları ilk sütunda, Veli'nin kazançları ise ikinci sütunda temsil edilmektedir. Her bir oyuncunun iki farklı stratejisi vardır: "A" veya "B".

Minimax'a göre: Ali, minimax stratejisini kullanarak karar verecektir. Onun amacı, en kötü durumu en aza indirmektir. Ali, kendi kazançlarını maksimize etmeye çalışırken Veli'nin en kötü tahminini yapar. Bu durumda, Ali'nin en iyi seçeneği "B" stratejisi olacaktır, çünkü Veli'nin "B" seçeneğinde daha düşük bir kazancı vardır. Ali, bu şekilde en kötü durumu en aza indirir.

Maximin'e göre: Veli, maximin stratejisini kullanarak karar verecektir. Onun amacı, kendisi için en iyi durumu garanti etmektir. Veli, kendi kazançlarını maksimize etmeye çalışırken Ali'nin en kötü tahminini yapar. Bu durumda, Veli'nin en iyi seçeneği "A" stratejisi olacaktır, çünkü Ali'nin "A" seçeneğinde daha düşük bir kazancı vardır. Veli, bu şekilde en kötü durumu en aza indirir.

Bu örnekte, minimax ve maximin stratejileri arasında bir denge yoktur. Ali'nin en iyi seçeneği "B" iken, Veli'nin en iyi seçeneği "A" dır. Bu durumda, oyuncular arasında çıkar çatışması vardır ve bir anlaşmaya varılmadığı sürece çıkmaza girilir. Eğer oyuncular arasında bir çıkmaz durumu oluşursa, yani minimax ve maximin stratejileri farklı sonuçlar veriyorsa, oyunun çözümü daha karmaşık hale gelir. Bu durumda oyuncular arasında uzlaşma veya alternatif çözüm yöntemleri kullanılabilir.

Bir çözüm bulabilmek için oyuncular arasında müzakereler yapılabilir veya oyun teorisinde kullanılan diğer çözüm kavramlarından biri uygulanabilir. Örneğin, Nash dengesi veya Pareto etkinliği gibi kavramlar, oyuncular arasında adil veya optimal bir çözüm sağlamaya yardımcı olabilir. Ayrıca, oyun teorisinde kullanılan bazı genişletilmiş çözüm kavramları da uygulanabilir. Örneğin, tekrarlayan oyunlarda oyuncular arasında iş birliği yapma veya cezalandırma stratejileri geliştirilebilir. Sonuç olarak, çıkmaz durumlarında oyuncuların iletişim kurması, müzakereler yapması ve alternatif çözüm yöntemlerini kullanması önemlidir. Oyun teorisi, oyuncular arasında etkileşimde bulunurken, daha adil, dengeleyici veya optimal sonuçlara ulaşmayı hedefleyen çözüm yöntemleri sunmaktadır.

3.2. Nash Dengesi

Nash dengesi, oyun teorisinde önemli bir kavramdır ve birçok stratejik oyunda istikrarlı bir çözüm noktası olarak kabul edilir. John Nash tarafından geliştirilen bu kavram, oyuncuların kendi stratejilerini optimize ettiğinde ortaya çıkan bir durumu ifade eder. Her oyuncunun stratejisinin, diğer oyuncuların stratejilerini göz önünde bulundurduğunda, değiştirmeye teşvik edilmediği bir noktayı ifade eder. Yani, hiçbir oyuncu tek başına kendi stratejisini değiştirerek daha iyi bir sonuç elde edemez (Bekmez ve Çalış, 2011).

Nash dengesi, oyuncular arasında birbirine bağımlı kararlar verildiği ve her oyuncunun diğer oyuncunun stratejilerini tahmin ettiği stratejik oyunlarda ortaya çıkar. Bu denge noktasında, oyuncuların kararlarının birbirini dikkate alarak alındığı ve hiçbir oyuncunun tek taraflı olarak avantaj elde etmediği bir durum söz konusudur (Myerson, 1999).

Örnek olarak, ünlü "Hapishane Mahkûmu" (Mahkumlar çıkmazı, Mahkumlar açmazı) oyununu ele alalım. Bu oyunda iki oyuncu, birbirlerine iş birliği yapma veya ihanet etme seçeneği sunulmuş bir durumdadır. Eğer her iki oyuncu da birbirine ihanet ederse, her ikisi de ceza alır. Eğer her iki oyuncu da iş birliği yaparsa, her ikisi de daha az ceza alır. Nash dengesi, her iki oyuncunun da birbirine ihanet ettiği durumu ifade eder. Hiçbir oyuncu tek başına stratejisini değiştirerek daha iyi bir sonuç elde edemez çünkü diğer oyuncunun davranışını tahmin etmiştir ve ihanet etmek avantaj sağlamaktadır.

Nash dengesi, çeşitli stratejik oyunlarda uygulanabilir ve oyuncular arasında istikrarlı bir çözüm noktası olarak kabul edilir. Ancak, Nash dengesi her zaman en iyi veya en optimal sonucu garanti etmez. Bazı durumlarda, oyuncular arasında daha iyi çözüm noktaları olabilir, ancak Nash dengesi oyuncuların birbirlerini tahmin ederek stratejilerini optimize ettikleri temel çözüm noktasını temsil eder.

Makine öğrenme algoritmaları da Nash dengesini analiz etmek ve keşfetmek için kullanılabilir. Özellikle, çok oyunculu oyunlarda ve stratejik karar verme problemlerinde makine öğrenme algoritmaları, Nash dengesini tahmin etmek ve çözmek için kullanılabilir (Razmi ve ark., 2020).

Bir örnek olarak, birçok oyunculu oyunu ele alalım. Diyelim ki birtakım oyuncular, belirli bir oyunun kurallarına göre farklı stratejiler izleyerek rekabet

ediyorlar. Bu oyuncuların her biri, kendi kazanç veya maliyet fonksiyonlarına sahiptir ve amacı bu fonksiyonu optimize etmektir.

Makine öğrenme algoritmaları, bu oyuncuların stratejilerini ve tercihlerini analiz ederek Nash dengesini tahmin etmeye yardımcı olabilir. Öncelikle, verilerin toplanması ve oyuncuların geçmiş stratejilerine ve performansına dayalı olarak bir öğrenme modeli oluşturulur.

Bu öğrenme modeli, oyuncuların stratejilerini tahmin etmek ve gelecekteki hamlelerini öngörmek için kullanılır. Ardından, model, oyuncuların stratejilerini optimize ederek kendi kazançlarını maksimize etmelerini sağlayacak yöntemleri belirlemeye çalışır.

Makine öğrenme algoritmaları, oyuncular arasındaki etkileşimleri ve strateji seçimlerini analiz ederek, Nash dengesini tahmin etmeye ve optimize etmeye çalışır. Bu algoritmaların kullanılmasıyla, oyuncular arasındaki rekabet ve etkileşimleri anlamak ve denge noktalarını keşfetmek mümkün olur.

Örneğin, birçok oyunculu oyunu analiz eden bir yapay zekâ algoritması, oyuncuların stratejilerini tahmin eder ve ardından bu tahminleri kullanarak oyunun potansiyel Nash dengelerini belirlemeye çalışır. Algoritma, oyuncuların stratejilerini ve davranışlarını iteratif olarak güncelleyerek, en iyi sonuçları elde etmeyi hedefler.

Makine öğrenme algoritmaları, geniş bir veri tabanı ve öğrenme süreci üzerinden Nash dengesini tahmin etme ve analiz etme kapasitesine sahiptir. Bu sayede, stratejik karar verme problemlerinde ve çok oyunculu oyunlarda oyuncular arasındaki etkileşimleri anlama ve denge noktalarını keşfetme konusunda yardımcı olabilirler.

Matematiksel olarak stratejik biçimli bir oyun;

$G = \{ N, (S_i), (u_i) \}$ Olarak tanımlandığında, eğer her i oyuncusunun s_i^* stratejisi diğer (n-1) oyuncunun $\left(s_1^*, \dots, s_{i-1,}^* s_{i+1,}^* \dots, s_n^* \right)$ stratejilerine en iyi tepkisi ise,

$s^* = (s_1^*, \dots s_n^*)$ Stratejisi tam strateji Nash dengesidir (Conitzer ve Sandholm, 2008). Bu durum aşağıdaki denklemde matematiksel olarak ifade edildi.

$$u_i \left(s_1^*, \dots, s_{i-1,}^* s_{i+1,}^* \dots, s_n^* \right) \geq u_i \left(s_1^*, \dots, s_{i-1,}^* s_{i+1,}^* \dots, s_n^* \right) \forall_i \in N, \forall s_i^* \in S_i$$

Burada s_i^*, i oyuncusunun faydasını maksimize eden değer olarak kabul edilir. Bu durumu aşağıdaki denklemde ifade ettim.

$s_i^* = max_{s_i \in S_i} u_i \left(s_1^*, \dots, s_{i-1}^*, s_{i+1}^*, \dots, s_n^* \right)$

Yukarıdaki denkleme göre, $G = \{ N, (S_i), (u_i) \}$ stratejik biçimli bir oyunun çözümünün (s_1', s_n') strateji profili olduğu varsayıldığında, eğer

(s_1', s_n') Strateji profilinin Nash dengesi değilse, i oyuncusu için $s_1'' \in S_i$ gibi daha yüksek fayda sağlayan başka bir strateji var demektir.

Örnek 5: 2 oyunculu bir oyunu Nash dengesi kavramıyla analiz etmek için bir Python kodu yazalım.

```
import numpy as np
def calculate_payoff(player1_action, player2_action):
    # Ödeme matrisini tanımla
    payoff_matrix = np.array([[3, 2], [1, 4]])
    # İlgili hücredeki ödeme değerini döndür
    return payoff_matrix[player1_action][player2_action]
def update_strategy(action, strategy, alpha):
    # Stratejiyi güncelleme
    strategy[action] += alpha * (1 - strategy[action])
    # Diğer stratejileri güncelleme
    for i in range(len(strategy)):
        if i != action:
            strategy[i] -= alpha * strategy[i]
            if strategy[i] < 0:
                strategy[i] = 0
    return strategy
def normalize_strategy(strategy):
    # Stratejiyi normalize etme
    strategy /= np.sum(strategy)
    return strategy
def play_game(iterations):
    # Oyuncuların başlangıç stratejileri
    player1_strategy = np.array([0.5, 0.5])
    player2_strategy = np.array([0.5, 0.5])
    for _ in range(iterations):
        # Stratejileri normalize etme
        player1_strategy = normalize_strategy(player1_strategy)
        player2_strategy = normalize_strategy(player2_strategy)
        # Oyuncuların eylemlerini rastgele seçme
        player1_action = np.random.choice([0, 1], p=player1_strategy)
        player2_action = np.random.choice([0, 1], p=player2_strategy)
```

```
        # Ödeme hesaplaması
        player1_payoff = calculate_payoff(player1_action, player2_action)
        player2_payoff = calculate_payoff(player2_action, player1_action)
        # Stratejileri güncelleme
        player1_strategy = update_strategy(player1_action, player1_strategy, player1_payoff)
        player2_strategy = update_strategy(player2_action, player2_strategy, player2_payoff)
    # Tahmini Nash dengesini yazdırma
    print("Player 1 Strategy:", player1_strategy)
    print("Player 2 Strategy:", player2_strategy)
# Oyunu oynatma
play_game(1000)
```

```
Player 1 Strategy: [1. 0.]
Player 2 Strategy: [0. 1.]
```

Şekil 7. İki oyunculu bir oyunda Nash dengesinin tahmini

Örnek 5'teki kod aşağıdaki adımları takip etmektedir:

1. ***calculate_payoff(player1_action, player2_action)*** fonksiyonu, oyuncuların eylemlerine (0 veya 1) bağlı olarak ödeme matrisinden ilgili ödeme değerini döndürmektedir. Ödeme matrisi payoff_Matris olarak tanımlanmıştır.
2. ***update_strategy(action, strategy, alpha)*** fonksiyonu, bir oyuncunun stratejisini güncellemektedir. action parametresi oyuncunun seçtiği eylemi temsil etmektedir. strategy parametresi ise oyuncunun mevcut stratejisini ifade etmektedir. alpha parametresi ise güncelleme faktörüdür. Fonksiyon, seçilen eyleme göre ilgili stratejiyi günceller ve diğer stratejileri de uygun şekilde ayarlar.
3. ***normalize_strategy(strategy)*** fonksiyonu, bir oyuncunun stratejisini normalize etmektedir. Bu, stratejideki olasılıkları toplamı 1 olacak şekilde düzenler.
4. ***play_game(iterations)*** fonksiyonu, oyunun belirli bir sayıda tekrarlanmasını sağlar. iterations parametresi ile kaç kez oynanacağı belirlenir. Oyuncuların başlangıç stratejileri eşit olarak belirlenir. Oyun döngüsü *'iterations'* kez tekrarlanır. Her iterasyonda oyuncuların stratejileri normalize edilir. Ardından oyuncular rastgele eylemler seçer ve ödeme hesaplaması yapılır. Ödeme değerlerine göre stratejiler güncellenir.
5. Oyunun sonunda, tahmini Nash dengesi olarak kabul edilen oyuncu stratejileri (player1_strategy ve player2_strategy) yazdırılır.

Kod, her iterasyonda oyuncuların eylemlerini rastgele seçiyor, ödemeleri hesaplıyor ve stratejilerini güncelliyor. Sonunda, tahmini Nash dengesini ekrana yazdırıyor.

Bu kod, belirli bir ödeme matrisi için iki oyuncunun birbirleriyle etkileşimini simüle eder. Oyuncular, başlangıçta eşit olasılıklı stratejilere sahiptir ve belirli bir sayıda iterasyon üzerinden oyunu oynarlar. Oyuncular, her turda stratejilerine dayanarak eylemlerini rastgele seçerler. Ardından, seçilen eylemler üzerinden ödeme hesaplanır ve stratejiler güncellenir. Strateji güncellemesi, oyuncunun seçtiği eylemin sonucuna bağlı olarak gerçekleşir.

Nash dengesini tahmin etmek için bir makine öğrenme algoritması olan Reinforcement Learning (Takviyeli Öğrenme) yöntemini kullanarak Python koduyla bir örnek yapalım. Ama öncesinde takviyeli ya da pekiştirmeli öğrenmeyi açıklamak isterim. Günlük yaşamda, bir insanın yoğun bir çalışma sürecinin ardından yorgunluk hissetmesi durumu ortaya çıkar. Bu durum, açlık ve dikkat seviyesi gibi iki parametreye bağlıdır. Bireyin vücudu, bu parametreleri değerlendirerek dinlenme ihtiyacını belirlemektedir. Benzer bir senaryoyu, robot süpürge için düşünebiliriz. Robot, bir sonraki odadaki pislikleri toplamak amacıyla şarj seviyesi ve oda ile olan mesafe gibi faktörleri göz önünde bulundurarak odayı temizlemek için bir karar verir. Bu örnekler, pekiştirmeli öğrenmeyi özetlemektedir. Örnek 6'da yazılan Python kodu, Reinforcement Learning algoritması kullanarak oyuncular arasındaki etkileşimleri modelleyerek ve stratejileri güncelleyerek Nash dengesini tahmin etmeye çalışır. Oyunu belirli bir sayıda iterasyonla oynatır ve her bir iterasyonda oyuncuların stratejilerini güncelleyerek öğrenme gerçekleştirir. Q-learning, Pekiştirmeli Öğrenme alanında öne çıkan bir algoritmadır ve bir ajanın çevresiyle etkileşimde bulunarak, ödül sinyallerini alarak ve deneyimlerini kullanarak en iyi eylem stratejisini öğrenmesini amaçlar. Bu, Pekiştirmeli Öğrenme alanının daha geniş kapsamı altında yer alır, burada odaklanılan nokta, bir ajanın belirli bir ortamda belirli hedefleri gerçekleştirmek için nasıl eylemler gerçekleştireceğini öğrenmesidir. Ajan, çevresinin durumlarını algılayarak, eylemler gerçekleştirerek ve ödül veya ceza sinyalleri alarak bu sinyalleri kullanarak en iyi eylem stratejisini öğrenir. Q-learning, bu öğrenme sürecini kolaylaştırmak için bir yöntem sunar ve ajanın eylemlerini değerlendirmek, güncellemek ve en iyi eylem stratejisini belirlemek için bir Q-matrisi kullanır. Bu şekilde, Q-learning, Pekiştirmeli

Öğrenme alanında güçlü bir algoritma olarak hizmet eder ve ajanın yinelemeli etkileşimler ve ödül sinyallerinin değerlendirilmesi yoluyla en etkili eylem stratejisini kazanmasını sağlar.

Örnek 6: Reinforcement Learning algoritması- Q Learning kullanarak Nash dengesini tahmin etmeye çalışalım.

```
import numpy as np
# Q-learning algoritması için gerekli parametreler
learning_rate = 0.1
discount_factor = 0.9
num_episodes = 100000
# Oyun durumu ve aksiyon sayıları
num_states = 3
num_actions = 2
# Q-table'ı başlangıçta rastgele değerlerle doldur
q_table = np.random.rand(num_states, num_actions)
# Oyuncu 1 için eylem seçimi
def select_action_player1(state):
    return np.argmax(q_table[state])
# Oyuncu 2 için eylem seçimi
def select_action_player2(state):
    return np.argmax(q_table[min(state, num_states - 1), :])
# Oyunda bir adım gerçekleştirme
def take_step(state_player1, state_player2):
    action_player1 = select_action_player1(state_player1)
    action_player2 = select_action_player2(state_player2)
    # Oyuncuların eylemleri üzerinden ödeme hesaplama
    if action_player1 == 0 and action_player2 == 0:
        payoff_player1 = 3
        payoff_player2 = 3
    elif action_player1 == 0 and action_player2 == 1:
        payoff_player1 = 1
        payoff_player2 = 4
    elif action_player1 == 1 and action_player2 == 0:
        payoff_player1 = 4
        payoff_player2 = 1
    else:
        payoff_player1 = 2
        payoff_player2 = 2
```

```
    # Q-table güncelleme
    q_table[state_player1, action_player1] += learning_rate * (payoff_player1 + discount_factor * np.max(q_table[state_player1])
    q_table[state_player2, action_player2] += learning_rate * (payoff_player2 + discount_factor * np.max(q_table[state_player2])
# Q-learning algoritmasıyla oyunu eğitme
for episode in range(num_episodes):
    state_player1 = np.random.randint(num_states)
    state_player2 = np.random.randint(num_states)
    take_step(state_player1, state_player2)
    # Her 1000 episode'de bir sonuçları yazdırma
    if (episode + 1) % 1000 == 0:
        strategy_player1 = np.argmax(q_table, axis=1)
        strategy_player2 = np.argmax(q_table, axis=0)
        print(f"Episode: {episode + 1}")
        print("Player 1 Strategy:", strategy_player1)
        print("Player 2 Strategy:", strategy_player2)
        print()
# Eğitim sonucunda Nash dengesini elde etmek için stratejileri belirleme
strategy_player1 = np.argmax(q_table, axis=1)
strategy_player2 = np.argmax(q_table, axis=0)
# Sonuçları yazdırma
print("Final Player 1 Strategy:", strategy_player1)
print("Final Player 2 Strategy:", strategy_player2)
```

```
Episode: 97000
Player 1 Strategy: [1 1 0]
Player 2 Strategy: [2 1]

Episode: 98000
Player 1 Strategy: [1 1 0]
Player 2 Strategy: [2 1]

Episode: 99000
Player 1 Strategy: [1 1 0]
Player 2 Strategy: [2 0]

Episode: 100000
Player 1 Strategy: [1 1 0]
Player 2 Strategy: [2 1]

Final Player 1 Strategy: [1 1 0]
Final Player 2 Strategy: [2 1]
```

Şekil 8. Q-learning algoritmasıyla iki oyunculu bir oyunu eğiterek Nash dengesini elde etme

Kodun işleyişi şu şekildedir:

- Başlangıçta, Q-table'ı rastgele değerlerle doldurulur. Q-table, her bir durum-aksiyon çifti için bir değer içeren bir tablodur. Tablodaki değerler, Q-learning algoritmasının öğrenme süreci boyunca güncellenecektir.
- ***select_action_player1(state)*** fonksiyonu, oyuncu 1 için bir aksiyon seçer. Bu aksiyon, Q-table'dan en yüksek değere sahip olan aksiyon olarak belirlenir.
- ***select_action_player2(state)*** fonksiyonu, oyuncu 2 için bir aksiyon seçer. Bu aksiyon, oyuncu 2'nin durumuna bağlı olarak Q-table'dan en yüksek değere sahip olan aksiyon olarak belirlenir.
- ***take_step(state_player1, state_player2)*** fonksiyonu, bir adım gerçekleştirir. Bu adımda, her iki oyuncunun seçtiği aksiyonlara bağlı olarak ödeme hesaplanır ve Q-table güncellenir. Ödeme değerleri, if-Gönül Su-else bloğu kullanılarak belirlenir.
- Q-learning algoritması, belirli sayıda tekrarlanan bölümler (episodes) üzerinden gerçekleştirilir. Her bir bölümde, başlangıç durumları

rastgele olarak seçilir ve ***take_step*** fonksiyonuyla bir adım gerçekleştirilir. Bu işlem, Q-table değerlerinin güncellenmesini sağlar.

- Her 1000 bölümde bir, güncellenen stratejiler yazdırılır. Oyuncu 1 ve oyuncu 2'nin stratejileri, Q-table'dan en yüksek değere sahip olan aksiyonlar olarak belirlenir.
- Eğitim sonunda, elde edilen Q-table'a dayanarak oyuncu 1 ve oyuncu 2'nin son stratejileri belirlenir ve yazdırılır.

Bu örnekte, iki oyunculu bir oyunda Q-learning algoritmasını kullanarak oyuncuların stratejilerini eğitiyoruz. Oyuncular, durumlarını temsil eden Q-table'ı kullanarak eylem seçiyor ve elde ettikleri ödeme değerlerine göre Q-table'ı güncelliyorlar. Sonunda, eğitim sonucunda elde edilen Q-table'ı kullanarak Nash dengesini elde etmek için her oyuncu, kendi stratejisini belirliyor. Bu stratejiler, Q-table'ın maksimum değerlere karşılık gelen aksiyonları seçerek belirleniyor. Oyuncular, bu stratejileri kullanarak Nash dengesine yakın bir oyun oynamaya çalışıyorlar. Q-learning algoritması kullanılarak eğitim yapıldıktan sonra elde edilen Q-table kullanılarak Nash dengesi yaklaşık olarak elde edilmeye çalışılmaktadır. Q-table'da her bir hücre, bir oyuncunun bir aksiyonu alması durumunda beklenen toplam ödeme değerini temsil etmektedir. Oyuncular, kendi stratejilerini belirlemek için Q-table'ın maksimum değerlere karşılık gelen aksiyonlarını seçerler. Nash dengesi, her bir oyuncunun kendi stratejisini maksimize ettiği ve diğer oyuncunun stratejisini değiştirmesi durumunda ekstra kazanç elde edemeyeceği bir noktayı temsil eder.

Bu örnekte, eğitim sonucunda elde edilen stratejiler, yaklaşık bir Nash dengesi oluşturmaya çalışmaktadır. Oyuncular, Q-table'da belirtilen stratejileri kullanarak birbirlerine karşı oyun oynamaktadırlar. Ancak, belirli bir oyun için tam bir Nash dengesi **garanti edilmemektedir.** Bu örnek, Q-learning algoritmasının Nash dengesi yaklaşımını kullanarak oyun stratejilerini öğrenebileceğini göstermektedir. Sonuçlar, eğitim parametreleri ve oyunun özelliklerine bağlı olarak değişebilir.

Nash dengesi, oyun teorisinde bir oyuncunun diğer oyuncuların stratejilerini göz önünde bulundurarak kendi en iyi stratejisini belirlediği noktadır. Nash dengesi, bir oyuncunun diğer oyuncuların stratejilerini

değiştirmeye teşvik etmediği ve dolayısıyla her oyuncunun kendi stratejik kararını en iyi şekilde aldığı bir noktayı ifade eder.

Formel olarak, bir oyunun Nash dengesi, her oyuncunun stratejik seçimlerini değiştirmek için bir teşvik bulunmadığı bir noktada gerçekleşir. Bu, her oyuncunun mevcut stratejileri altında maksimum kazanç elde ettiği bir durumu ifade eder. Nash dengesi, bireysel oyuncuların rasyonel davrandığı ve diğer oyuncuların stratejik tepkilerini dikkate aldığı varsayımına dayanır. Bu varsayıma göre, oyuncular kendi karlarını maksimize etmek için hareket ederler ve diğer oyuncuların stratejilerini tahmin ederek en iyi stratejilerini belirlerler.

Nash dengesi, bir oyunun istikrarlı bir çözüm noktasını temsil eder. Oyuncular, Nash dengesindeki stratejilerini değiştirmeyi tercih etmezler, çünkü bu değişiklik onların kazançlarını azaltabilir. Ancak, her oyuncunun birbirinin stratejilerini tam olarak tahmin etmesi veya rasyonel bir şekilde hareket etmesi durumunda, Nash dengesi garantilenemez. Nash dengesi, bir oyunun analizinde önemli bir kavramdır ve stratejik etkileşimlerin sonuçlarını anlamak için kullanılır. Bu kavram, birçok farklı oyun teorisi uygulamasında kullanılan bir temel kavramdır. Bir oyunda baskın strateji dengesi varsa bu aynı zamanda sofistike dengedir ve her sofistike dengede bir Nash dengesidir.

John Nash'in "Non-Cooperative Games" adlı makalesi, oyun teorisinde önemli bir adım olarak kabul edilen ve Nash denge kavramının temelini oluşturan bir çalışmadır. Makalenin özeti şu şekildedir (Nash, 1951):

Nash, çalışmasında rekabetçi olmayan oyunlara odaklanır, yani oyuncuların birbirleriyle iş birliği yapmadığı durumları inceler. Bu tür oyunlarda her oyuncu, diğer oyuncuların stratejilerini dikkate alarak kendi stratejik kararını verir. Makalede, Nash, her oyuncunun kendi stratejisini belirlemek için bir fayda fonksiyonu kullandığını varsayar. Fayda fonksiyonları, oyuncuların belirli stratejiler altında elde ettikleri kazançları temsil eder. Nash, bir oyuncunun en iyi stratejisini belirlemek için diğer oyuncuların stratejik tepkilerini dikkate alması gerektiğini gösterir. Bu şekilde, her oyuncunun stratejik kararını değiştirmesi için bir teşvik bulunmaz. Yani, bir oyuncu, diğer oyuncuların stratejilerini tahmin ederek kendi stratejisini seçtiğinde, herhangi bir stratejik değişiklik yapmaya teşvik edilmez. Nash, bu tür bir stratejik karar noktasını "Nash dengesi" olarak adlandırır. Nash dengesi, her oyuncunun mevcut stratejileri altında maksimum kazancını elde ettiği bir durumu ifade eder. Makalede, Nash ayrıca, oyunun herhangi bir durumunda en

az bir Nash dengesi olduğunu ve bazı durumlarda birden fazla Nash dengesi olabileceğini gösterir. Bu durum, oyuncuların farklı stratejileri tercih edebileceği ve farklı Nash dengelerine ulaşılabileceği anlamına gelir.

Nash'in makalesi, oyun teorisine önemli bir katkı yaparak rekabetçi olmayan oyunlarda dengeli stratejik karar noktalarının varlığını gösterir. Bu çalışma, Nash dengesinin temelini oluşturarak oyun teorisi alanında büyük ilgi uyandırmış ve birçok uygulama ve analizin temelini oluşturmuştur.

Bir oyunun Nash dengesi, her oyuncunun mevcut stratejisi altında diğer oyuncuların stratejilerini değiştirmeye teşvik etmediği bir noktadır. Nash'in ispatı, bu dengenin varlığını göstermek için özellikle sabit nokta teoremini kullanır. Sabit nokta teoremi, bir işlevin sabit noktasının her zaman var olduğunu gösterir. Oyun teorisi bağlamında, bu teorem, bir oyunun her bir oyuncusu için bir strateji profili olduğunu ve bu strateji profili altında hiçbir oyuncunun stratejisini değiştirmek istemediği bir noktanın her zaman bulunduğunu belirtir.

Nash'in ispatı, bir oyunun her bir oyuncusu için bir fayda fonksiyonu kullanır. Bu fonksiyonlar, oyuncuların stratejilerine ve diğer oyuncuların stratejilerine bağlı olarak oyuncuların faydalarını hesaplar. İspat, oyuncuların fayda fonksiyonlarına dayalı olarak birbirlerinin stratejilerini tahmin edebildiklerini ve en iyi stratejilerini seçtiklerini varsayar (Fudenberg ve Tirole, 1991).

İspatta, her oyuncu kendi faydasını maksimize etmeye çalışırken diğer oyuncuların stratejilerini de dikkate aldığı varsayılır. Bu durumda, oyuncuların stratejik tepkilerini ve en iyi stratejilerini belirlemeleri için matematiksel analiz yapılır. Nash'in teoreminin tam matematiksel ispatı oldukça karmaşık ve teknik bir konu olup, matematiksel notasyon ve sembollerle ifade edilir. Bu kitapta buna yer verilmemiştir. Merak edenler için John Nash'in 1950'de yayımlanan orijinal makalesinin tam ismi "Non-Cooperative Games"dir. Bu makale, Nash'in oyun teorisi üzerine yaptığı temel çalışmalardan biridir ve Nash denge kavramını tanıtarak oyun teorisinde devrim niteliğinde bir ilerleme sağlamıştır. Makale, "Annals of Mathematics" dergisinin 2. serisi, 54. cildinde yayımlanmıştır (Nash, 1951).

Başka bir örnekte Nash dengesini bulmaya çalışalım. İki şirket arasındaki rekabet durumunda fiyat stratejilerini ele alarak oluşturalım. Şirket A ve Şirket B, düşük fiyat (D) ve yüksek fiyat (Y) olmak üzere iki farklı

stratejiyi tercih edebilirler. Matrisin her bir hücresindeki rakamlar, her iki şirketin belirli bir strateji kombinasyonunu uyguladığında elde edeceği kârları temsil eder.

		ŞİRKET B	
		Düşük Fiyat	Yükek Fiyat
ŞİRKET A	Düşük Fiyat	3 , 3	1, 4
	Yüksek Fiyat	4 , 1	2 , 2

Bu matriste, her bir hücredeki değerler, Şirket A ve Şirket B'nin belirli bir fiyat stratejisi kombinasyonunu uyguladığında elde edeceği kârları temsil eder. Örneğin, (3, 3) hücresi, Şirket A'nın düşük fiyat stratejisini ve Şirket B'nin düşük fiyat stratejisini uyguladığı durumda elde edilecek kârı gösterir.

Şimdi, Nash dengesini bulmak için her bir şirketin stratejik tercihlerini değerlendirelim. ***Nash dengesi, hiçbir şirketin tek başına stratejisini değiştirerek daha fazla kâr elde etmesine müsaade etmez.*** Böylelikle en kârlı sonucun iki şirketinde düşük fiyat politikasını tercih ettiği durumda oluştuğu görülmektedir.

İki kişinin ya da kurumunda Nash dengesi sağlanırken seçimlerinde karlı çıktıkları durumlarda olabilir. Diyelim ki iki ev arkadaşı aynı arabayı kullanmak için mücadele ediyor. Her biri arabayı kullanma zamanı konusunda kendi tercihlerine sahip.

Ev Arkadaşı A, işe gitmek için sabah saatlerinde arabayı kullanmak istiyor. Ev Arkadaşı B ise akşam saatlerinde sosyal etkinliklere katılmak için arabayı kullanmak istiyor. Her iki ev arkadaşı kendi çıkarlarını maksimize etmeye çalışırken, Nash denge noktası, her iki arkadaşın da arabayı eşit bir şekilde kullanmayı kabul ettiği bir durumdur. Sonuç olarak, Ev Arkadaşı A arabayı sabah kullanacak ve Ev Arkadaşı B ise akşam kullanacak şekilde bir

anlaşmaya varabilirler. Her ikisi de kendi çıkarlarını maksimize ederken, arabayı eşit bir şekilde paylaşma noktasında buluşurlar ve Nash denge noktası sağlanır.

Örnek 7: Matris üzerinde Nash dengesini bulmaya çalışan Python kodunu yazalım ve daha sonra kodun ne yaptığını açıklayalım.

```python
import numpy as np
# Kar matrisi (örnek değerler)
matrix = np.array([[3, 2], [0, 5]])
# Oyuncu sayısı ve strateji sayısı
num_players = len(matrix)
num_strategies = len(matrix[0])
# Her oyuncu için maksimum kazancı tutacak bir dizi
max_payoffs = np.zeros(num_players)
# Her oyuncu için maksimum kazanca ulaşacak stratejileri tutacak bir dizi
best_strategies = np.zeros(num_players, dtype=int)
# Her oyuncunun stratejilerini döngüyle kontrol etme
for player in range(num_players):
    max_payoff = np.max(matrix[player])
    max_payoffs[player] = max_payoff
    best_strategy = np.argmax(matrix[player])
    best_strategies[player] = best_strategy
# Nash dengesini kontrol etme
nash_equilibria = np.where(max_payoffs == np.max(max_payoffs))[0]
# Sonuçları yazdırma
if len(nash_equilibria) > 0:
    for player in range(num_players):
        print(f"Oyuncu {player+1}'in stratejisi: {best_strategies[player]}")
else:
    print("Nash dengesi bulunamadı.")
# Tüm stratejileri yazdırma
for player in range(num_players):
    print(f"Oyuncu {player+1}'in stratejileri: {list(range(num_strategies))}")
```

```
Oyuncu 1'in stratejisi: 0
Oyuncu 2'in stratejisi: 1
Oyuncu 1'in stratejileri: [0, 1]
Oyuncu 2'in stratejileri: [0, 1]
```

Şekil 9. Matris üzerinde Nash dengesini bulmaya çalışan Python kodu ve Çıktısı

Bu kod, bir matris üzerinde Nash dengesini bulmayı amaçlayan bir Python programıdır. İşleyişini adım adım açıklayalım:

- Kâr matrisi (Matrix) örnek değerlerle tanımlanır. Bu matris, oyuncuların stratejilerine ve kazançlarına karşılık gelen değerleri içerir.

- Oyuncu sayısı (num_players) ve strateji sayısı (num_strategies) hesaplanır.
- Her oyuncu için maksimum kazancı (max_payoffs) tutacak bir dizi oluşturulur.
- Her oyuncu için maksimum kazanca ulaşacak stratejileri (best_strategies) tutacak bir dizi oluşturulur.
- Her oyuncunun stratejileri döngüyle kontrol edilir. Oyuncunun en yüksek kazanca ulaşacağı strateji belirlenir ve ilgili dizilere kaydedilir.
- Nash dengesini kontrol etmek için, maksimum kazancı paylaşan oyuncuların indeksleri (nash_equilibria) bulunur.
- Eğer en az bir Nash dengesi bulunmuşsa, her oyuncunun ilgili stratejisi yazdırılır. Aksi takdirde, "Nash dengesi bulunamadı" mesajı verilir.
- Son olarak, her oyuncunun tüm stratejileri yazdırılır.

Bu kod, verilen matris üzerinde Nash dengesini bulmaktadır. Her oyuncu, maksimum kazanca ulaşacağı stratejiyi seçer ve bu stratejilerin tüm oyuncular arasında paylaşılmasıyla Nash dengesi elde edilir.

Makine öğrenme algoritmaları genellikle veri analizi, modelleme ve tahminleme gibi görevler için kullanılırken, ***Nash dengesi gibi oyun teorisi kavramlarını hesaplamak için doğrudan bir makine öğrenme yaklaşımı kullanmak gerekli değildir.*** Nash dengesi, verilen bir oyunda oyuncuların stratejilerinin birbirine en iyi yanıtlarını bulmayı amaçlayan bir matematiksel kavramdır.

Örneğin, verilen bir oyun matrisine dayanarak Nash dengesini hesaplamak için optimize edilmiş matematiksel veya algoritmik yaklaşımlar kullanmak daha uygun olabilir. Nash dengesi genellikle doğrusal programlama, matris oyun teorisi veya diğer optimizasyon teknikleriyle bulunur.

Ancak, makine öğrenme algoritmaları, oyun teorisi problemlerinin bazı yönlerini çözmede yardımcı olabilir. Örneğin, önceden belirlenmiş bir oyun matrisi yerine gerçek veriyle çalışılması gereken dinamik veya belirsiz oyun senaryolarında, makine öğrenme algoritmaları kullanılabilir. Bu durumlarda, algoritma, verilere dayanarak oyuncuların stratejilerini öğrenme veya tahmin etme amacıyla kullanılabilir. Yani, makine öğrenme algoritmalarının doğrudan Nash dengesi hesaplaması için gerekli olmadığı, ancak oyun teorisi problemlerinin belirli yönlerinde kullanılabileceği söylenebilir.

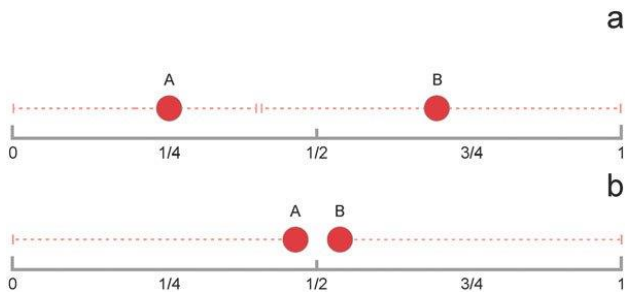
Harold Hotelling, 1929 yılında yayınladığı "Stability in Competition" adlı makalesinde bu rekabet modelini tanıtmıştır. Hotelling, modelinde iki benzer ürün satıcısının aynı çizgide yer aldığını ve müşterilerin yerel olarak tercih ettikleri ürüne göre hareket ettiklerini varsaymıştır.

Hotelling rekabetinde satıcılar, müşterilere daha yakın olmak için yer seçiminde rekabet ederler. Ancak, rekabetin sonucunda satıcılar birbirlerine yaklaşma eğiliminde olurlar ve sonuçta tam ortaya yerleşme, yani Nash dengesi ortaya çıkar.

Bu rekabet modeli, birçok sektörde uygulanabilir. Örnek olarak, restoranlar, benzin istasyonları, mağazalar gibi yerel pazarlarda rekabet eden işletmeler arasındaki konum seçimine ilişkin davranışları açıklamada kullanılabilir.

Örneğin, bir şehirdeki iki pizza restoranı düşünelim. Şehir, 1 km uzunluğunda bir cadde üzerinde yer alıyor ve iki restoran da bu cadde üzerinde konumlanmak istiyor. Restoranlar, müşterilere benzer kalitede pizzalar sunuyor ve müşterilerin hareket etme maliyeti ihmal edilebilir düzeyde. Restoranlar, stratejilerini cadde üzerindeki konumlarını seçerek belirler. Eğer her iki restoran da aynı konuma yerleşirse, müşteriler aralarında tercih yapmak zorunda kalmadan herhangi bir restorana gidebilirler.

Bu durumda, Nash dengesi, her iki restoranın da eşit mesafede yarı yarıya bölerek cadde üzerinde konumlanmaları olabilir. Örneğin, restoranlardan biri caddeyi 0,52 km noktasında yer alırsa, diğer restoran onun hemen yanına açarak 0,51 km noktasında yer alır. Diğer restorana 0,49 km'lik yer kalacağı için bunu kabul etmez çünkü kârı düşer. Yer değiştirerek 0,50 km noktasında yer alır. Diğer firmada aynı konumda yer alır. Sonuçta tam ortaya yerleşerek Nash dengesini sağlamış olurlar. Nash dengesine göre, restoranlar bu konumları seçerek maksimum kârı elde ederler. Herhangi bir restoran tek taraflı olarak konumunu değiştirirse, müşterilerin tercihleri doğrultusunda diğer restoranın da stratejisini güncellemesi beklenir. Böylece denge noktası değişebilir.



Şekil 10. Hotelling rekabeti

Hotelling rekabeti örneği, benzer veya homojen ürünlere sahip olan pazarlarda rekabetin nasıl şekillenebileceğini ve oyuncuların stratejilerini nasıl belirleyeceğini anlamak için kullanılan bir modeldir (Adler, Brudner ve Proost, 2021). Bu model, pazar analizi ve strateji oluşturma süreçlerinde kullanılarak rekabetçi avantaj elde etmek amacıyla yapılan analizlerde faydalı olabilir.

3.3. Pareto Etkinliği

Pareto etkinliği ve oyun kuramı, farklı perspektiflerden kaynakların dağılımı ve karar verme süreçleriyle ilgilenir. İkisi arasında bir ilişki kurabiliriz.

Oyun kuramı, oyuncuların kararlarını optimize etmeye çalıştığı etkileşimli bir model sağlar. Oyuncuların stratejileri, kararlarının sonuçlarına ve diğer oyuncuların stratejilerine bağlı olarak belirlenir. Oyun kuramı, oyuncuların rasyonel davrandığı ve kendi çıkarlarını maksimize etmeye çalıştığı varsayımına dayanır.

Pareto etkinliği ise bir durumun, en az bir oyuncunun durumunu iyileştirirken diğer oyuncunun durumunu kötüleştirmeyen bir duruma ulaşmasıdır. Pareto etkinliği, bir kaynak dağılımının veya bir durumun herkes için optimal olduğu durumu ifade eder. Oyun kuramı içinde, oyuncuların kararları ve stratejileri, genellikle kaynakların paylaşılması, rekabet veya iş birliği gibi durumlarla ilişkilendirilir. Pareto etkinliği, bu oyunların bir sonucu olarak ortaya çıkar. Bir oyuncunun stratejisinin değişmesi veya bir oyunun sonucunda Pareto İyileştirmesi yapılması durumunda, Pareto etkinliği artar. Yani, bir oyuncunun durumunu iyileştirirken diğer oyuncuların durumu kötüleştirilmemiştir (Aydın ve Karabacak, 2023).

Örneğin, iki oyunculu bir oyunu ele alalım. Oyuncular arasında bir kaynak veya kazanç paylaşımı söz konusu olsun. Oyuncuların stratejileri, kendi

kazançlarını maksimize etmeyi hedefleyebilir. Pareto etkinliği, bu oyunun sonucunda ortaya çıkar. Eğer bir strateji değişikliği veya iş birliği ile bir oyuncunun kazancı arttırılırken, diğer oyuncunun kazancı düşürülmezse, Pareto İyileştirmesi gerçekleşir ve Pareto etkinliği sağlanır.

Pareto etkinliği ve oyun kuramı arasındaki ilişki, kaynakların ve kararların adil ve etkin bir şekilde dağıtılmasıyla ilgili önemli bir konuyu vurgular. Oyun kuramı, oyuncular arasındaki etkileşimleri modelleyerek optimal stratejileri analiz ederken, Pareto etkinliği, bu stratejilerin sonuçlarını değerlendirir ve adil ve optimal çözümlerin varlığını araştırır.

Aşağıdaki örnek, makine öğrenme algoritmaları ve oyun kuramını kullanarak Pareto etkinliğini göstermektedir. Örnek, iki oyunculu bir oyunda Nash dengesini bulmayı ve Pareto etkinliğini sağlamayı hedeflemektedir.

Örnek 8: Pareto etkinliğini ve oyuncu strateji kazançlarını kontrol eden Python kodunu yazalım.

```
import numpy as np

def calculate_payoff(player1_action, player2_action):
    payoff_matrix = np.array([[4, 1], [3, 2]])
    return payoff_matrix[player1_action][player2_action]

def find_nash_equilibrium(payoff_matrix):
    num_actions = payoff_matrix.shape[0]

    # Oyuncu 1 için Nash dengesi stratejisi
    player1_strategy = np.zeros(num_actions)
    max_payoff = float('-inf')
    for action in range(num_actions):
        expected_payoff = np.max(payoff_matrix[action, :])
        if expected_payoff > max_payoff:
            max_payoff = expected_payoff
            player1_strategy = np.zeros(num_actions)
            player1_strategy[action] = 1
        elif expected_payoff == max_payoff:
            player1_strategy[action] = 1

    # Oyuncu 2 için Nash dengesi stratejisi
    player2_strategy = np.zeros(num_actions)
    max_payoff = float('-inf')
    for action in range(num_actions):
        expected_payoff = np.max(payoff_matrix[:, action])
        if expected_payoff > max_payoff:
            max_payoff = expected_payoff
            player2_strategy = np.zeros(num_actions)
            player2_strategy[action] = 1
        elif expected_payoff == max_payoff:
            player2_strategy[action] = 1

    return player1_strategy, player2_strategy

def check_pareto_efficiency(payoff_matrix, player1_strategy, player2_strategy):
    num_actions = payoff_matrix.shape[0]
```

```
    for action1 in range(num_actions):
        payoff1 = np.dot(payoff_matrix[action1, :], player2_strategy)
        for action2 in range(num_actions):
            payoff2 = np.dot(payoff_matrix[:, action2], player1_strategy)
            if payoff1 < payoff_matrix[action1, action2] and payoff2 < payoff_matrix[action1, action2]:
                return False
    return True

# Ödeme matrisi
payoff_matrix = np.array([[4, 1], [3, 2]])

# Nash denge stratejilerini bulma
player1_strategy, player2_strategy = find_nash_equilibrium(payoff_matrix)
print("Player 1 Nash Strategy:", player1_strategy)
print("Player 2 Nash Strategy:", player2_strategy)

# Pareto etkinliğini kontrol etme
pareto_efficient = check_pareto_efficiency(payoff_matrix, player1_strategy, player2_strategy)

if pareto_efficient:
    print("The outcome is Pareto efficient.")
else:
    print("The outcome is not Pareto efficient.")

# Ödeme matrisini kontrol etme
print("Payoff Matrix:")
print(payoff_matrix)

# Stratejileri kontrol etme
print("Player 1 Strategy Payoffs:")
for action1 in range(payoff_matrix.shape[0]):
    payoff1 = np.dot(payoff_matrix[action1, :], player2_strategy)
    print("Action", action1, "Payoff:", payoff1)
```

```
print("Player 2 Strategy Payoffs:")
for action2 in range(payoff_matrix.shape[0]):
    payoff2 = np.dot(payoff_matrix[:, action2], player1_strategy)
    print("Action", action2, "Payoff:", payoff2)
```

```
Player 1 Nash Strategy: [1. 0.]
Player 2 Nash Strategy: [1. 0.]
The outcome is Pareto efficient.
Payoff Matrix:
[[4 1]
 [3 2]]
Player 1 Strategy Payoffs:
Action 0 Payoff: 4.0
Action 1 Payoff: 3.0
Player 2 Strategy Payoffs:
Action 0 Payoff: 4.0
Action 1 Payoff: 1.0
```

Şekil 11. Pareto etkinliği, Nash dengesi ve oyuncu strateji kazançları

Bu kod, ödeme matrisi ve oyuncu stratejileri üzerinde çeşitli hesaplamalar yaparak Nash denge stratejilerini, Pareto etkinliğini ve oyuncu strateji kazançlarını kontrol eden bir oyun teorisi örneğidir.

İşleyişi şu şekildedir:

1. ***calculate_payoff*** fonksiyonu, oyuncuların eylemlerine (actions) dayalı olarak ödeme matrisindeki ödeme (payoff) değerlerini hesaplar.
2. ***find_nash_equilibrium*** fonksiyonu, verilen ödeme matrisi üzerindeki oyuncu stratejilerini hesaplar. Oyuncu 1 için, her bir eylem için

beklenen ödemenin en yüksek olduğu stratejiyi belirler. Aynı şekilde, oyuncu 2 için de en yüksek beklenen ödeme stratejilerini belirler.

3. ***check_pareto_efficiency*** fonksiyonu, verilen ödeme matrisi ve oyuncu stratejileri üzerinde Pareto etkinliğini kontrol eder. Pareto etkinlik, herhangi bir oyuncunun ödemesini artırırken diğer oyuncunun ödemesini azaltmadığı bir durumu ifade eder. Bu fonksiyon, tüm eylem çiftlerini kontrol ederek Pareto etkinliğini kontrol eder.

4. Verilen ödeme matrisi üzerinde Nash denge stratejilerini ***(find_nash_equilibrium)***, Pareto etkinliğini ***(check_pareto_efficiency)*** ve oyuncu strateji kazançlarını (calculate_payoff) kontrol eden kodlar yer almaktadır.

Örnek 9: Makine öğrenme algoritmalarından birini kullanarak Pareto etkinliğini açıklayan başka bir örnek yapalım.

```
import numpy as np
from sklearn.linear_model import LinearRegression
# Örnek veri seti
X = np.array([[1, 2], [2, 4], [3, 6], [4, 8], [5, 10]])
Y = np.array([2, 4, 6, 8, 10])
# Lineer regresyon modelini eğitme
model = LinearRegression()
model.fit(X, Y)
# Modelin tahminlerini al
predictions = model.predict(X)

# Pareto etkinliği kontrolü
pareto_efficient_indices = []
for i in range(len(predictions)):
    is_pareto_efficient = True
    for j in range(len(predictions)):
        if predictions[j] > predictions[i]:
            is_pareto_efficient = False
            break
    if is_pareto_efficient:
        pareto_efficient_indices.append(i)
# Pareto etkin sonuçları yazdırma
print("Pareto etkin sonuçlar:")
for index in pareto_efficient_indices:
    print("X:", X[index], "Y:", Y[index])
```

```
Pareto etkin sonuçlar:
X: [ 5 10] Y: 10
```

Şekil 12. Veri seti üzerinde bir lineer regresyon modeli eğiterek, model tahminleme ile pareto etkinliğinin kontrol edilmesi.

Bu Python kodu, örnek bir veri seti üzerinde bir lineer regresyon modeli eğitiyor ve ardından bu modelin tahminlerini kullanarak Pareto etkinliğini kontrol ediyor. İlk olarak, ***X*** ve ***Y*** adında bir örnek veri seti oluşturuluyor. ***X*** veri seti, özelliklerin (bağımsız değişkenlerin) bir dizisini içerirken, Y veri seti hedef değişkeni (bağımlı değişkeni) içerir.

Daha sonra, ***LinearRegression*** sınıfı kullanılarak bir lineer regresyon modeli oluşturuluyor ve ***fit*** yöntemi kullanılarak model veri setine uyduruluyor. Model, veri setindeki ilişkiyi temsil eden bir doğruya uyum sağlamaya çalışır.

Sonraki adımda, ***predict*** yöntemi kullanılarak modelin veri seti üzerindeki tahminleri elde edilir. Bu, veri setindeki her bir örnek için bir tahmin değeri döndürür.

Daha sonra, Pareto etkinliğini kontrol etmek için bir döngü kullanılır. Her bir örneğin tahmin değeri diğer tüm örneklerin tahmin değerleriyle karşılaştırılır. Eğer başka bir örneğin tahmin değeri, incelenen örneğin tahmin değerinden büyükse, o zaman incelenen örnek Pareto etkin değildir ve döngüden çıkılır. Eğer hiçbir örneğin tahmin değeri, incelenen örneğin tahmin değerinden büyük değilse, o zaman incelenen örnek Pareto etkin olarak kabul edilir ve indeks listesine eklenir.

Son olarak, Pareto etkin olan örneklerin indeksleri kullanılarak orijinal veri setindeki ilgili özellik ve hedef değerleri yazdırılır.

3.4. Köşegen Oyunlar

Oyun kuramında köşegen oyunlar, oyuncuların aynı stratejiyi seçtiği veya oyuncuların her birinin stratejileri farklı olsa bile sonuçların köşegen (diyagonal) oluştuğu oyunlardır. Köşegen oyunlarda, her oyuncunun en iyi hamlesi, diğer oyuncunun hamlesine bağlı değildir.

Bir örnek üzerinden köşegen oyunları açıklayalım.

Örneğin, A ve B adında iki oyuncu arasında oynanan bir köşegen oyuna bakalım. Her oyuncunun iki farklı stratejisi vardır: "Kazan" (W) ve "Kaybet" (L). Aşağıda oyuncuların ödeme matrisini görelim:

		A	
		W	L
	W	2, 2	1, 3
B			
	L	3, 1	0, 0

Bu matriste, her bir hücrede (oyuncu B'nin stratejisi, oyuncu A'nın stratejisi) şeklinde ödeme değerleri bulunmaktadır. Örneğin, (W, W) stratejisi durumunda A oyuncusu 2 birim ödeme alırken, B oyuncusu da 2 birim ödeme alır.

Bu köşegen oyunda, her bir oyuncunun köşegen stratejisi (W, W) olduğunda en yüksek ödemeyi alacağı görülür. Diğer bir deyişle, her iki oyuncu da "Kazan" stratejisini seçtiğinde Pareto etkin bir durum elde edilir. Diğer tüm durumlarda (L, L) ve (W, L) stratejilerinde oyuncuların ödemeleri daha düşüktür.

Köşegen oyunlar, oyuncular arasında iş birliği veya rekabetin belirgin olmadığı, her bir oyuncunun kendi çıkarını en üst düzeye çıkarmayı hedeflediği durumları temsil eder. Oyuncuların strateji seçimi, diğer oyuncunun stratejilerine bağlı olmadığından, köşegen oyunlar daha basit bir yapıya sahiptir ve analitik çözümlere daha elverişlidir.

Örnek 10: Köşegen oyunlarını Python kodu kullanarak gösterebiliriz.

```python
import numpy as np
def solve_zero_sum_game(payoff_matrix):
    num_rows, num_cols = payoff_matrix.shape
    # Sütunların toplamını bulma
    col_totals = np.sum(payoff_matrix, axis=0)
    # Oyuncu A'nın stratejisi (kazanma olasılıkları)
    player_a_strategy = col_totals / np.sum(col_totals)
    # Oyuncu B'nin stratejisi (kaybetme olasılıkları)
    player_b_strategy = 1 - player_a_strategy
    # Oyuncu A'nın beklenen kazançları
    expected_payoffs_a = np.dot(payoff_matrix, player_b_strategy)

    # Oyuncu B'nin beklenen kazançları
    expected_payoffs_b = np.dot(payoff_matrix.T, player_a_strategy)
    # Nash dengesindeki maksimum beklenen kazanç
    max_expected_payoff = np.max(expected_payoffs_a)
    # Nash denge stratejileri
    nash_equilibrium_a = np.where(expected_payoffs_a == max_expected_payoff)[0]
    nash_equilibrium_b = np.where(expected_payoffs_b == max_expected_payoff)[0]
    return player_a_strategy, player_b_strategy, nash_equilibrium_a, nash_equilibrium_b
# Köşegen oyunun ödeme matrisi
payoff_matrix = np.array([[2, 1], [3, 0]])
# Köşegen oyununu çözme
player_a_strategy, player_b_strategy, nash_equilibrium_a, nash_equilibrium_b = solve_zero_sum_game(payoff_matrix)
print("Oyuncu A'nın Stratejileri (Kazanma Olasılıkları):", player_a_strategy)
print("Oyuncu B'nin Stratejileri (Kaybetme Olasılıkları):", player_b_strategy)
print("Oyuncu A'nın Nash Dengesi Stratejileri:", nash_equilibrium_a)
print("Oyuncu B'nin Nash Dengesi Stratejileri:", nash_equilibrium_b)
```

```
Oyuncu A'nın Stratejileri (Kazanma Olasılıkları): [0.83333333 0.16666667]
Oyuncu B'nin Stratejileri (Kaybetme Olasılıkları): [0.16666667 0.83333333]
Oyuncu A'nın Nash Dengesi Stratejileri: [0]
Oyuncu B'nin Nash Dengesi Stratejileri: []
```

Şekil 13. Köşegen Matris kullanımı ile Kazanma olasılıklarının hesaplanması

Köşegen matrisde, herhangi bir oyuncunun kazancı, diğer oyuncunun kaybıdır. Bu kod, bir sıfır toplamlı oyunun ödeme matrisini kullanarak oyunu çözmek için bir işlev içerir. İlk olarak, ***solve_zero_sum_game*** adında bir işlev tanımlanır ve ödeme matrisi parametre olarak alınır. Daha sonra, ödeme matrisinin satır ve sütun sayıları alınır. ***col_totals*** adında bir değişken oluşturulur ve ödeme matrisinin sütunlarının toplamını içerir. Bu, her bir stratejinin olasılığını temsil eder. ***player_a_strategy*** adında bir değişken oluşturulur ve ***col_totals*** değişkeninin toplamına bölünerek oyuncu A'nın stratejisi (kazanma olasılıkları) elde edilir. ***player_b_strategy*** adında bir değişken oluşturulur ve 1'den player_a_strategy çıkarılarak oyuncu B'nin stratejisi (kaybetme olasılıkları) elde edilir. ***expected_payoffs_a*** adında bir değişken oluşturulur ve ödeme matrisi ile ***player_b_strategy*** vektörü arasında nokta çarpımı yaparak oyuncu A'nın beklenen kazançlarını hesaplar. ***expected_payoffs_b*** adında bir değişken oluşturulur ve ödeme matrisinin transpozunu (transpose) alarak oyuncu B'nin beklenen kazançlarını hesaplar. ***max_expected_payoff*** adında bir değişken oluşturulur ve ***expected_payoffs_a*** vektöründeki maksimum beklenen kazancı bulur. ***nash_equilibrium_a*** adında bir değişken oluşturulur ve ***expected_payoffs_a*** vektöründe maksimum

beklenen kazanca sahip olan stratejilerin indekslerini içerir. ***nash_equilibrium_b*** adında bir değişken oluşturulur ve ***expected_payoffs_b*** vektöründe maksimum beklenen kazanca sahip olan stratejilerin indekslerini içerir. Son olarak, bu değerler ***return*** ifadesiyle döndürülür.

Kodun çıktısı, ödeme matrisine bağlı olarak değişebilir. Oyuncu A'nın stratejileri (kazanma olasılıkları), oyuncu B'nin stratejileri (kaybetme olasılıkları) ve Nash dengesi stratejileri (maksimum beklenen kazanca sahip olan stratejiler) ekrana yazdırılır. Eğer ödeme matrisi herhangi bir Nash dengesine sahip değilse, çıktı boş olabilir.

Makine öğrenme algoritmaları, genellikle büyük veri setleriyle ilişkilendirilen karmaşık örüntüleri öğrenme ve tahmin yapma amacıyla kullanılır. Ancak, oyun kuramı ve köşegen oyunları gibi konulara doğru bir şekilde yaklaşmak için Python veya diğer programlama dillerinde analitik metotlar ve matematiksel formüller kullanılması daha yaygındır. Bu yüzden köşegen oyunları için makine öğrenme algoritmalarından herhangi birisinin kullanıldığı bir örnek paylaşmayacağım.

3.5. Simetrik Oyunlar

Simetrik oyunlar, oyun kuramında önemli bir kavramdır. Bir oyunun simetrik olması, oyuncuların stratejileri ve ödemeleri açısından birbirine eşit veya benzer olduğu anlamına gelir. Simetrik oyunlar, oyuncuların aynı stratejik seçeneklere sahip olduğu ve her oyuncunun aynı ödemeler matrisine sahip olduğu oyunlardır (Smith, 1979). Örneğin, Taş-Kâğıt-Makas oyunu simetrik bir oyundur. Her iki oyuncu da taş, kâğıt ve makas gibi aynı üç stratejik seçeneğe sahiptir ve herhangi bir oyuncunun taşı, kâğıdı veya makası seçmesi durumunda ödemeler matrisi aynıdır. Bu tür bir oyunun ödeme matrisi genellikle simetrik olarak gösterilir.

Simetrik oyunlar, oyun kuramında analiz edilirken önemli avantajlara sahiptir. Çünkü oyuncuların benzer stratejik durumları olduğu için analitik çözümler daha kolay bulunabilir. Simetrik oyunlarda, Nash dengesi gibi önemli kavramlar daha hızlı ve doğrudan tespit edilebilir (Hammerstein ve Selten, 1994).

Simetrik oyunlar, stratejik etkileşimleri ve dengeleri anlamak için oyun kuramını kullanarak analiz edilebilir. Oyuncuların stratejik seçimlerini

modellemek için matematiksel formüller ve ödeme matrisleri kullanılır. Simetrik oyunlarda, oyuncuların birbirine karşı aynı stratejik durumda oldukları için karşılıklı olarak benzer hareketleri takip etmeleri beklenir. Python veya diğer programlama dilleri, simetrik oyunların analitik çözümünü hesaplamak ve ödeme matrisini temsil etmek için kullanılabilir. Matris hesaplamaları, Nash dengelerini bulmak veya oyunun diğer özelliklerini analiz etmek için yapılabilecek işlemler arasındadır.

Örnek 11: Makine öğrenme algoritmalarını simetrik oyunlarla ilişkilendirebiliriz. Simetrik bir oyunda makine öğrenme algoritması kullanarak Python koduyla yazdığım bir örnek:

```
import numpy as np
# Oyuncu sayısı
num_players = 2
# Oyuncuların stratejik seçenekleri
actions = ['A', 'B', 'C']
# Oyuncuların stratejilerini rastgele başlatma
player_strategies = []
for _ in range(num_players):
    player_strategies.append(np.random.choice(actions))
# Oyuncuların ödeme matrisini oluşturma
payoff_matrix = np.array([[3, 2, 1], [1, 3, 2], [2, 1, 3]])
# Makine öğrenme algoritması (Q-Learning)
Q = np.zeros((len(actions), len(actions)))
def calculate_payoff(strategies):
    # Oyuncuların stratejilerini kullanarak ödemeleri hesapla
    payoffs = []
    for player in range(num_players):
        strategy_indices = [actions.index(strategies[p])
                            for p in range(num_players) if p != player]
        payoff = np.dot(payoff_matrix[strategy_indices]
                        [:, player], [actions.index(strategies[player])])
        payoffs.append(payoff)
    return payoffs
def choose_action(strategy):
    # Epsilon-Greedy yöntemiyle bir aksiyon seçme
    if np.random.uniform(0, 1) < epsilon:
        return np.random.choice(actions)
    else:
        return actions[np.argmax(Q[strategy])]
# Parametreler
epsilon = 0.2  # Keşif olasılığı
alpha = 0.1  # Öğrenme hızı
gamma = 0.9  # İndirim faktörü
```

```
# Oyunu oynatma
for iteration in range(10):
    # Oyuncuların stratejilerini yazdırma
    print("Iteration", iteration + 1)
    for player in range(num_players):
        print("Player", player + 1, "Strategy:", player_strategies[player])
    # Oyuncuların ödemelerini hesaplama
    payoffs = calculate_payoff(player_strategies)
    for player in range(num_players):
        print("Player", player + 1, "Payoff:", payoffs[player])
    # Oyuncuların stratejilerini güncelleme (Q-Learning)
    for player in range(num_players):
        current_strategy = actions.index(player_strategies[player])
        action = choose_action(current_strategy)
        action_index = actions.index(action)
        Q[current_strategy][action_index] = (1 - alpha) * Q[current_strategy][action_index] + alpha * (
                    payoff_matrix[current_strategy][action_index] + gamma * np.max(Q[action_index]))
        player_strategies[player] = action
```

Şekil 14. Köşegen Matris kullanımı ile Kazanma olasılıklarının hesaplanması

Bu kod, iki oyunculu bir oyunda Q-Learning algoritmasını kullanarak oyuncuların stratejilerini güncellemek için bir örnek sunmaktadır. İşleyiş aşağıdaki adımlarla gerçekleştirilir:

1. Oyuncu sayısı ***(num_players)*** ve stratejik seçenekler ***(actions)*** belirlenir.
2. Oyuncuların başlangıç stratejileri rastgele seçilir ve ***player_strategies*** listesine kaydedilir.
3. Ödeme matrisi ***(payoff_Matris)*** tanımlanır. Bu matris, her bir oyuncunun her bir strateji kombinasyonunda elde edeceği ödemeleri içerir.
4. Q matrisi başlangıçta sıfırlardan oluşacak şekilde tanımlanır. Bu matris, her bir oyuncunun strateji çiftlerine ilişkin değerleri tutar.
5. ***calculate_payoff*** fonksiyonu, oyuncuların mevcut stratejilerine göre ödemeleri hesaplar.
6. ***choose_action*** fonksiyonu, epsilon-greedy yöntemini kullanarak bir aksiyon seçer.
7. Parametreler ***(epsilon, alpha, gamma)*** belirlenir. Epsilon, keşif olasılığını; alpha, öğrenme hızını; gamma, indirim faktörünü temsil eder.
8. Belirli bir sayıda iterasyon için aşağıdaki adımlar tekrarlanır:
    - Oyuncuların mevcut stratejileri yazdırılır.
    - ***calculate***_payoff fonksiyonuyla oyuncuların ödemeleri hesaplanır ve yazdırılır.
    - Oyuncuların stratejileri güncellenir. Bu adımda Q-Learning algoritması kullanılır. Her oyuncu için sırasıyla aşağıdaki işlemler yapılır:

- Mevcut strateji ve aksiyon seçimi alınır.
- Q matrisi güncellenir.
- Oyuncunun stratejisi, seçilen aksiyonla değiştirilir.

Bu şekilde, her iterasyonda oyuncuların stratejileri güncellenerek Q-Learning algoritması kullanılarak oyun oynanır.

Bu örnekte, Q-Learning adlı bir makine öğrenme algoritması kullanılarak simetrik bir oyunda stratejiler öğrenilmeye çalışılıyor. Oyuncuların stratejik seçenekleri, ödeme matrisi ve makine öğrenme algoritması parametreleri tanımlanıyor. Ardından, her iterasyonda oyuncuların stratejileri ve ödemeleri hesaplanıyor. Makine öğrenme algoritması Q-Learning algoritmasının güncelleme adımlarıyla oyuncuların stratejileri öğrenilmeye çalışılıyor. Oyunun her bir iterasyonunda oyuncuların stratejileri ve ödemeleri yazdırılıyor.

Bu örnekte, Q-Learning kullanılarak oyuncuların stratejileri iterasyonlarla güncellenir. Her bir oyuncu, epsilon-greedy yöntemiyle aksiyon seçer. Yani bazen rastgele bir aksiyon seçerken bazen de Q değerlerine dayanarak en yüksek değere sahip aksiyonu seçer. Q değerleri, her adımda güncellenerek oyuncuların stratejilerini iyileştirir.

Bu şekilde, simetrik bir oyunda makine öğrenme algoritmaları kullanarak oyuncuların stratejilerini optimize etmek mümkündür. Oyunun ilerleyen iterasyonlarında, oyuncuların stratejileri istikrarlı bir şekilde güncellenerek Nash dengesine yaklaşabilirler.

Örnek 12: Genetik Algoritma kullanarak simetrik bir oyunda stratejileri optimize etmek için bir Python kodu yazalım.

```python
import random
# Sabitler
population_size = 10
num_iterations = 50
gene_length = 2
mutation_rate = 0.1
# Hedef fonksiyon
def fitness_function(strategy):
    return strategy[0] + 2 * strategy[1]
# Başlangıç popülasyonunu oluşturma
def generate_initial_population():
    population = []
    for _ in range(population_size):
        strategy = [random.randint(0, 5) for _ in range(gene_length)]
        population.append(strategy)
    return population
# Popülasyonu değerlendirme
def evaluate_population(population):
    fitness_scores = []
    for strategy in population:
        fitness_scores.append(fitness_function(strategy))
    return fitness_scores
# Çaprazlama
def crossover(parent1, parent2):
    crossover_point = random.randint(1, gene_length - 1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2
# Mutasyon
def mutation(strategy):
    for i in range(gene_length):
        if random.random() < mutation_rate:
            strategy[i] = random.randint(0, 5)
    return strategy
# Başlangıç popülasyonunu oluşturma
population = generate_initial_population()
for iteration in range(num_iterations):
    # Popülasyonu değerlendirme
    fitness_scores = evaluate_population(population)
    # En iyi stratejiyi ve ödemesini bulma
    best_strategy = population[fitness_scores.index(max(fitness_scores))]
    best_payoff = max(fitness_scores)
    # En iyi stratejiyi ve ödemesini yazdırma
    print(f"Iteration {iteration+1} - Best Strategy: {best_strategy}, Best Payoff: {best_payoff}")
    # Yeni neslin oluşturulması
    next_generation = []
    while len(next_generation) < population_size:
        # Ebeveyn seçimi
        parent1 = random.choice(population)
        parent2 = random.choice(population)
        # Çaprazlama
        child1, child2 = crossover(parent1, parent2)
        # Mutasyon
        child1 = mutation(child1)
        child2 = mutation(child2)
        # Yeni nesile ekleme
        next_generation.append(child1)
        next_generation.append(child2)
    # Sonraki nesilin popülasyonunu güncelleme
    population = next_generation
```

Şekil 15. Genetik algoritma kullanarak bir optimizasyon problemi

Bu kod, genetik algoritma kullanarak bir optimizasyon problemi çözmektedir. Aşağıda kodun yaptığı işlemleri açıklamaya çalışayım:

1. ***population_size, num_iterations, gene_length*** gibi bazı sabitler tanımlanır. Bu sabitler, genetik algoritmanın çalışması için gerekli olan parametreleri belirtir.
2. ***fitness_function*** adında bir fonksiyon tanımlanır. Bu fonksiyon, bir stratejinin (genotipin) uygunluk puanını hesaplar. Bu örnekte, uygunluk puanı, stratejinin ilk geniyle ikinci geninin iki katının toplamıdır.

fitness_function(strategy) = strategy[0] + 2 * strategy[1] şeklindedir.

3. ***generate_initial_population*** adında bir fonksiyon tanımlanır. Bu fonksiyon, başlangıç popülasyonunu oluşturur. Her bir strateji, gene_length uzunluğunda rastgele sayılardan oluşan bir dizi olarak temsil edilir.
4. ***evaluate_population*** adında bir fonksiyon tanımlanır. Bu fonksiyon, popülasyonun uygunluk puanlarını hesaplar. Her strateji için uygunluk puanı, fitness_function fonksiyonu kullanılarak hesaplanır.
5. ***crossover*** adında bir fonksiyon tanımlanır. Bu fonksiyon, iki ebeveyn stratejisi arasında çaprazlama (crossover) işlemi yapar. Çaprazlama noktası rastgele seçilir ve yeni çocuk stratejileri oluşturulur.
6. ***mutation*** adında bir fonksiyon tanımlanır. Bu fonksiyon, bir stratejide mutasyon (mutation) yapar. Her gen için, ***mutation_rate*** olasılığına bağlı olarak rastgele bir sayıyla değiştirilir.
7. Başlangıç popülasyonu oluşturulur ve her bir iterasyonda aşağıdaki adımlar tekrarlanır:
    - Popülasyon değerlendirilir ve en iyi strateji ile ödemesi bulunur.
    - Yeni nesil stratejiler oluşturulur. Ebeveynler seçilir, çaprazlama yapılır ve mutasyon uygulanır.
    - Sonraki nesil popülasyonu güncellenir.
    - En iyi strateji ve ödeme yazdırılır.

Bu şekilde genetik algoritma iterasyonlarla çalışır ve her bir iterasyonda popülasyonun uygunluk puanlarına göre en iyi stratejiyi ve ödemesini günceller.

Bu örnekte, Genetik Algoritma kullanılarak simetrik bir oyunda oyuncuların stratejileri optimize edilmeye çalışılıyor. Popülasyon başlangıçta

rastgele stratejilerle oluşturuluyor. Her iterasyonda popülasyon değerlendiriliyor ve en iyi strateji seçiliyor. Daha sonra, çaprazlama ve mutasyon operatörleri kullanılarak yeni nesil oluşturuluyor. Bu işlem, belirli bir sayıda iterasyon boyunca tekrarlanıyor. Bu kod, simetrik bir oyunda Genetik Algoritma kullanarak oyuncuların stratejilerini optimize etmeyi amaçlamaktadır.

3.6. Muhalif Oyunlar

Muhalif oyunlar, oyun kuramı kavramlarından biridir. Muhalif oyunlar, birden fazla oyuncunun rekabet ettiği ve bir oyuncunun kazancının diğer oyuncuların kaybına eşit olduğu oyunlardır. Bu tür oyunlarda oyuncuların hedefleri çatışır ve bir oyuncunun kazanması diğerlerinin kaybetmesiyle sonuçlanır.

Muhalif oyunlar genellikle sıfır toplamlı oyunlar olarak da adlandırılır, çünkü oyuncuların toplam kazancı veya kaybı sıfıra eşittir. Bu tür oyunlarda, bir oyuncunun kazancı diğer oyuncuların kaybına tam olarak denk gelir, yani toplam kazanç miktarı sabittir.

Muhalif oyunlar, stratejik düşünme, rekabet ve optimal karar verme becerilerini gerektirir. Örnek olarak, satranç, poker, ticaret müzakereleri gibi birçok oyunda muhalif oyunlarla karşılaşabilirsiniz.

Örnek 13: Genetik Algoritma kullanarak simetrik bir oyunda stratejileri optimize etmek için bir Python kodu yazalım.

```python
import random
# Tahta boyutu
BOARD_SIZE = 3
# Oyun tahtası
board = [[' '] * BOARD_SIZE for _ in range(BOARD_SIZE)]
# Kedi ve köpek sembolleri
CAT_SYMBOL = 'C'
DOG_SYMBOL = 'D'
# Kedinin hamlesini gerçekleştirir
def cat_move():
    best_score = float('-inf')
    best_move = None
    # Tahta üzerindeki boş hücreleri dener
    for i in range(BOARD_SIZE):
        for j in range(BOARD_SIZE):
            if board[i][j] == ' ':
                board[i][j] = CAT_SYMBOL
                score = minimax(board, 0, False)
                board[i][j] = ' '
                # En iyi hamleyi seçer
                if score > best_score:
                    best_score = score
                    best_move = (i, j)
    # Kedinin en iyi hamlesini gerçekleştirir
    if best_move:
        board[best_move[0]][best_move[1]] = CAT_SYMBOL
# Köpeğin hamlesini gerçekleştirir
def dog_move():
    available_moves = []
    # Boş hücreleri bulur
    for i in range(BOARD_SIZE):
        for j in range(BOARD_SIZE):
            if board[i][j] == ' ':
                available_moves.append((i, j))
    # Köpeğin rastgele hamlesini gerçekleştirir
    if available_moves:
        move = random.choice(available_moves)
        board[move[0]][move[1]] = DOG_SYMBOL
# Tahta üzerindeki kazananı kontrol eder
def check_winner():
    for i in range(BOARD_SIZE):
        # Satırları kontrol eder
        if board[i][0] == board[i][1] == board[i][2] != ' ':
            return board[i][0]
        # Sütunları kontrol eder
        if board[0][i] == board[1][i] == board[2][i] != ' ':
            return board[0][i]
    # Çaprazları kontrol eder
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return board[0][2]
    # Berabere durumunu kontrol eder
    if all(board[i][j] != ' ' for i in range(BOARD_SIZE) for j in range(BOARD_SIZE)):
        return 'Berabere'
    return None
```

```
# Minimax algoritması
def minimax(board, depth, is_maximizing):
    winner = check_winner()
    if winner:
        if winner == CAT_SYMBOL:
            return 1
        elif winner ==  DOG_SYMBOL:
            return -1
        else:
            return 0
    if is_maximizing:
        best_score = float('-inf')
        for i in range(BOARD_SIZE):
            for j in range(BOARD_SIZE):
                if board[i][j] == ' ':
                    board[i][j] = CAT_SYMBOL
                    score = minimax(board, depth + 1, False)
                    board[i][j] = ' '
                    best_score = max(score, best_score)
        return best_score
    else:
        best_score = float('inf')
        for i in range(BOARD_SIZE):
            for j in range(BOARD_SIZE):
                if board[i][j] == ' ':
                    board[i][j] = DOG_SYMBOL
                    score = minimax(board, depth + 1, True)
                    board[i][j] = ' '
                    best_score = min(score, best_score)
        return best_score
# Oyun döngüsü
def play_game():
    current_player = CAT_SYMBOL
    while True:
        if current_player == CAT_SYMBOL:
            cat_move()
            current_player = DOG_SYMBOL
        else:
            dog_move()
            current_player = CAT_SYMBOL
        print_board()
        winner = check_winner()
        if winner:
            if winner == 'Berabere':
                print("Oyun berabere bitti.")
            else:
                print("Kazanan:", winner)
            break
# Tahtayı yazdırır
def print_board():
    for row in board:
        print('|'.join(row))
        print('-' * (BOARD_SIZE * 2 - 1))
# Oyunu başlatır
def start_game():
    print("Kedi ve Köpek Oyununa Hoş Geldiniz!")
    print("Tahta Boyutu:", BOARD_SIZE, "x", BOARD_SIZE)
    print("Kedi sembolü:", CAT_SYMBOL)
    print("Köpek sembolü:", DOG_SYMBOL)
    print("Kedi oyuna başlıyor.")
    print_board()
    play_game()
# Oyunu başlatır
start_game()
```

Şekil 16. Kedi – Köpek Oyunu

Bu Python kodunda, Minimax algoritması kullanılarak kedi ve köpek arasındaki bir tahta oyunu simüle edilmiştir. Kedinin amacı, mümkün olan en yüksek skora ulaşmaktır, köpeğin ise mümkün olan en düşük skora ulaşmaktır. Algoritma, rekürsif (bir fonksiyonun kendi kendini çağırdığı programlama yöntemi) olarak tüm olası hamleleri değerlendirir ve en iyi hamleyi seçer. Bu kod, "Kedi ve Köpek" adlı bir oyunu simüle etmektedir. Aşağıda kodun yaptığı işlemleri açıklamaya çalışayım:

1. ***BOARD_SIZE*** adında bir sabit tanımlanır ve oyun tahtasının boyutunu belirler. Bu örnekte, tahta 3x3'lük bir kare matris olarak temsil edilmektedir.

2. ***board*** adında bir boş tahta oluşturulur. Bu tahta, başlangıçta her hücreyi boş bir karakterle (' ') doldurur.

3. ***CAT_SYMBOL ve DOG_SYMBOL*** adında sembolik değerler tanımlanır. Bu semboller, kedinin ve köpeğin tahta üzerindeki temsilini belirtir.

4. ***cat_move*** adında bir fonksiyon tanımlanır. Bu fonksiyon, kedinin hamlesini gerçekleştirir. Kedinin en iyi hamleyi bulmak için Minimax algoritmasını kullanır. Tahta üzerindeki boş hücreleri dener, her bir hamleyi yapar ve Minimax algoritması ile hamlenin puanını hesaplar. En iyi puanı ve hamleyi seçer ve kedinin tahta üzerindeki en iyi hamlesini gerçekleştirir.

5. ***dog_move*** adında bir fonksiyon tanımlanır. Bu fonksiyon, köpeğin hamlesini rastgele olarak gerçekleştirir. Tahta üzerindeki boş hücreleri bulur ve rastgele bir hamle seçer.

6. ***check_winner*** adında bir fonksiyon tanımlanır. Bu fonksiyon, tahta üzerindeki kazananı kontrol eder. Satırları, sütunları ve çaprazları tarar, aynı sembole sahip olan hücreleri kontrol eder ve bir kazanan varsa sembolünü döndürür. Ayrıca, tahta dolu ise berabere durumunu da kontrol eder.

7. ***minimax*** adında bir fonksiyon tanımlanır. Bu fonksiyon, Minimax algoritmasını uygular. Minimax, tüm hamleleri ağaç şeklinde arayarak en iyi hamleyi bulmaya çalışan bir rekürsif bir algoritmadır. Fonksiyon, tahta durumunu, derinliği ve sıranın kimde olduğunu (en yüksek puanı almak için maksimize eden oyuncu veya en düşük puanı almak için minimize eden oyuncu) parametre olarak alır.

8. ***play_game*** adında bir fonksiyon tanımlanır. Bu fonksiyon, oyunun ana döngüsünü gerçekleştirir. Oyuncular sırayla hamle yaparlar ve tahta

durumu ve kazananı kontrol ederler. Oyun sona erdiğinde kazanan veya berabere durumu ekrana yazdırılır.

9. ***print_board*** adında bir fonksiyon tanımlanır. Bu fonksiyon, tahtayı ekrana yazdırır. Tahta üzerindeki her hücrenin değerini '|' karakteriyle ayrılarak ve '-' karakteriyle çizgiler ekleyerek gösterir.

10. ***start_game*** adında bir fonksiyon tanımlanır. Bu fonksiyon, oyunu başlatır.

Hatırlatma: Bu örnek, basit bir tahta oyununu temsil etmek için verilmiştir. Daha karmaşık muhalif oyunlarda farklı yaklaşımlar ve algoritmalar kullanılabilir. Ayrıca, bu kod örneği sadece bir başlangıç noktasıdır ve iyileştirilmeye açıktır.

Bu örnekte, muhalif oyunlarla ilgili birkaç önemli noktaya değinebiliriz:

1. *Strateji:* Muhalif oyunlarda oyuncuların belirli stratejileri vardır ve bu stratejilere göre hamle yaparlar. Stratejiler, oyuncuların kazanma ihtimallerini artırmak veya rakibi etkisiz hale getirmek için kullanılan taktiklerdir.

2. *Oyuncu Seçimi:* Oyuncular, kendilerine en uygun stratejileri seçerler. Bu seçim, genellikle oyuncuların kazanma ihtimallerini maksimize etmek veya rakibe karşı üstünlük sağlamak amacıyla yapılır.

3. *Karar Verme:* Muhalif oyunlarda oyuncular, hamlelerini yaparken karar verme sürecine girerler. Karar verme, mevcut oyun durumunu, stratejilerini ve rakibin hareketlerini analiz ederek en iyi hamleyi seçme sürecidir.

4. *Kazananı Belirleme:* Muhalif oyunlarda kazanan, genellikle belirli bir kural setine veya duruma bağlı olarak belirlenir. Örneğin, taş-kağıt-makas oyununda, taş kağıdı kırar, kağıt taşı sarar ve makas kağıdı keser. Bu kurallara göre hareket eden oyuncu, oyunu kazanır.

5. *Strateji Geliştirme:* Oyuncular, muhalif oyunlarda stratejilerini geliştirebilir. Bu, deneyim, öğrenme ve analiz yoluyla gerçekleştirilebilir. Makine öğrenmesi algoritmaları kullanarak, oyuncuların stratejileri güncellenerek daha iyi sonuçlar elde edilebilir.

3.7. Mahkumlar Açmazı ve Baskın Strateji Dengesi

Bu senaryoda, bireylerin kararları birbirleriyle bağlantılıdır ve karşılıklı bir etkileşim söz konusudur. Eğer benim kaderim bir başkasına bağlıysa veya başkasının kaderi bana bağlıysa, bu durumda kararımı verirken dikkate alacağım faktörler arasında karşılıklı güven eksikliği, risk ve kendi çıkarlarım yer alacaktır (Kealey ve Ricketts, 2014).

Mahkûm ikilemi senaryosunda, benim iş birliği yapmak veya ihanet etmek gibi iki seçeneğim bulunurken, diğer kişi de aynı seçeneklerle karşı karşıyadır. Kararımı verirken, diğer kişinin davranışını tahmin etmeye çalışırım ve kendi çıkarlarımı en üst düzeyde tutmaya çalışırım.

Eğer karşı tarafın da benimle aynı şekilde düşünerek kendi çıkarlarını gözetmesi ve iş birliği yapmayı tercih etmesi bekleniyorsa, ben de iş birliği yaparak kazan-kazan sonucunu elde etmeyi hedeflerim. Ancak, karşı tarafın ihanet etmesini veya benim iş birliği yapmamı beklememesini tahmin ediyorsam, benim çıkarlarımı korumak adına ihanet etmek daha cazip gelebilir. Doğal olarak kararımı verirken diğer kişinin hareketlerini, beklentilerimi ve çıkarlarımı dikkate alırım. Mahkûm ikilemi senaryosunda, her iki tarafın da kendi çıkarlarını gözetmesi ve karşılıklı güvenin olmaması nedeniyle genellikle her iki taraf da ihanet etmeyi tercih eder ve sonuçta kaybeden-kazanan (kaybet-kazan) bir durum ortaya çıkar.

Aşağıdaki matrise bedel matrisi denilmektedir. A her mahkumların iş birliği yapması durumundaki kazançlarını, D mahkumların inkâr etmesi durumunda alacakları cezayı, B ve C ise mahkumlar farklı davrandıklarında ödeyecekleri bedeli göstermektedir.

		Mahkûm 2	
		İtiraf	İnkâr
Mahkûm 1	İtiraf	A	B
	İnkâr	C	D

Mahkumlar açmazında, C>A>D>B sıralaması esastır. Her iki mahkûm içinde en iyi strateji evrimsel kararlı strateji olarak ifade edilen inkâr etmektir. İnkâr etmenin cazibesi C>A ve aldatılma korkusu B<D iş birliğini riske atar.

Bu senaryo, suçu ispatlanamayan mahkumları etkileşim olmadan, kendi çıkarlarını maksimize etmelerini zorlaştıran bir psikolojik ve stratejik durumu modellemektedir. Bu senaryoyu açıklayarak mahkumlar açmazını daha iyi anlamaya çalışalım.

Senaryo: Suç işledikleri gerekçesiyle gözaltına alınan iki kişi var ve bu iki kişinin suçlu olduğuna dair herhangi bir kanıt yok! Kanıt olmamasına rağmen gözaltında tutulan iki birey ayrı odalara yerleştirilir ve polisler, bu bireylere ikilemde kalacakları bir teklif sunar. Birbirlerinden habersiz bir şekilde kaderlerini değiştirecek kararı vermek zorundalar.

a) İkisi de suçlarını itiraf ederse beşer yıl hapis cezasına çarptırılacaklar!
b) Eğer ikisi de sessiz kalıp bir şey söylemezse birer yıl hapis yatacaklar!
c) Biri itiraf eder, diğeri sessiz kalırsa sessiz kalan 10 yıl hapis yatacak, itiraf eden ise serbest bırakılacak!

Oyun teorisinde faydaları bir matrisle ifade ediyoruz. Fayda matrisi, oyuncuların stratejilerini analiz etmek, stratejik tercihlerini değerlendirmek ve olası sonuçları tahmin etmek için kullanılır. Oyun teorisinde, bu matrisler aracılığıyla çeşitli matematiksel yöntemler kullanılarak en iyi stratejileri, denge

noktalarını veya kazanç dağılımlarını belirlemek mümkün olur. Bu şekilde, oyuncuların kararlarını analiz etmek ve stratejik sonuçları anlamak için fayda matrisleri oyun teorisinde önemli bir araçtır. Fayda matrisi;

		Mahkûm 2	
		İtiraf	İnkâr
Mahkûm 1	İtiraf	5 Yıl , 5 Yıl	Serbest , 10 Yıl
	İnkâr	10 Yıl , Serbest	1 Yıl , 1 Yıl

		Mahkûm 2	
		İtiraf	İnkâr
Mahkûm 1	İtiraf	-5 , -5	0 , -10
	İnkâr	-10 , 0	-1 , -1

Fayda matrisinde inkâr ya da itirafta ceza alacakları için durumu negatif olarak ifade ederiz. Mavi renkle ifade ettiklerimiz mahkûm 1 için ve kırmızı ile ifade ettiklerimiz ise mahkûm 2 için yazılanları temsil etmektedir. Yazılanları rakamsal olarak ifade ettikten sonra oyunun çözümüne geçelim.

Rasyonel kararların alınacağı düşünülerek karşılaşılacak strateji baskın strateji dengesi olacaktır. Bu durumda şunu söyleyebiliriz;

Mahkûm 1 için: Mahkûm 2 ne yaparsa ne derse desin itiraf etmeyi seçecektir. İtiraf Mahkûm 1 için baskın stratejidir. Mahkûm 1 şöyle düşünür. Ben itiraf edersem Mahkûm 2 inkâr ederse serbest kalırım. Ben itiraf edersem ve Mahkûm 2'de itiraf ederse 10 yıl yerine 5 yıl ceza alırım. İki durumda da daha az ceza alacağım!

Mahkûm 2 için: Mahkûm 1 ne yaparsa ne derse desin itiraf etmeyi seçecektir. İtiraf Mahkûm 2 için baskın stratejidir. Mahkûm 2 şöyle düşünür. Ben itiraf edersem Mahkûm 1 inkâr ederse serbest kalırım. Ben itiraf edersem ve Mahkûm 1'de itiraf ederse 10 yıl yerine 5 yıl ceza alırım. İki durumda da daha az ceza alacağım!

Doğal olarak şunu söyleyebiliriz ki, (itiraf, itiraf) oyunun baskın stratejisi dengesidir. Tabi ki iki mahkûmun birbiriyle iletişim kuramaması bu sonucu doğurmaktadır. İletişim halinde olabilselerdi oyunun mahkumlar açısından en iyi sonucu 1'er yıl hapis alacakları (inkâr, inkâr) seçeneği olacaktı.

		Mahkûm 2	
		İtiraf	İnkâr
Mahkûm 1	İtiraf	5 Yıl , 5 Yıl	Serbest , 10 Yıl
	İnkâr	10 Yıl , Serbest	1 Yıl , 1 Yıl

Pareto optimal durum, bir tür "kazan-kazan" durumunu ifade eder.

(İtiraf, İtiraf) durumu pareto optimal değildir. (İnkâr, İnkâr) ise pareto optimaldir çünkü her iki oyuncuya da daha fazla fayda getiren durumu ifade etmektedir.

Mahkumlar açmazı günlük hayatımızda karşımıza birçok şekilde çıkabilir. İki rakip firma veya işletme arasında rekabet etmek zorunda olan durumlarda da mahkumlar açmazı yaşanabilir. Her iki firma da birbirine karşı rekabet ederek kârlarını artırmak isteyebilir. Ancak her biri agresif fiyat

indirimleri veya diğer rekabetçi taktikler uyguladığında, sonuçta her iki firma da zarar görebilir. İş birliği yapma veya sürdürülebilir rekabet stratejileri uygulamama konusunda bir ikilem ortaya çıkabilir. Yine başka bir durumda bireylerin çevreye olan etkileri ve sürdürülebilirlik arasında bir ikilem yaşanır. Bir kişi enerji tasarruflu cihazlar kullanarak veya geri dönüşüm yaparak çevreyi koruma amacıyla davranırsa, kısa vadede ek maliyetlerle karşılaşabilir. Ancak bu davranış toplumda yaygınlaştığında ve herkes benzer çabalar gösterirse, uzun vadede çevresel sorunları hafifletebilir. Tüketiciler, daha ucuz fiyata sahip olan ve kısa vadeli çıkar sağlayabilecek ürünleri seçme arasında bir ikilem yaşayabilir. Daha ucuz bir ürün seçmek, kısa vadede tasarruf sağlayabilirken, uzun vadede daha kaliteli ve dayanıklı bir ürün seçmek daha mantıklı olabilir. Bir grup projesinde çalışan bireyler arasında, her biri kendi çıkarlarını koruma veya grup başarısı için iş birliği yapma arasında bir ikilem yaşanabilir. Bireyler kendi çalışmalarına odaklanarak daha fazla övgü veya tanınma elde etmek isteyebilir. Ancak grup üyeleri birbirleriyle iş birliği yaparak bilgi paylaşımı yaparsa, sonuçta proje daha başarılı ve tatmin edici bir şekilde tamamlanabilir.

3.8. Sofistike Denge

"Sofistike denge" (sophisticated equilibrium), oyun teorisi terimlerinden biridir ve bir stratejik oyunun istikrarlı bir denge noktasını ifade eder (Camerer, 1991). Sofistike denge, oyuncuların kararlarını, diğer oyuncuların kararlarını da dikkate alarak rasyonel bir şekilde yapmalarını gerektirir. Sofistike denge, bir oyunda oyuncuların en iyi tepki stratejilerini kullanarak birbirlerine yanıt verdikleri bir noktayı ifade eder. Bu denge noktasında, oyuncuların stratejilerini değiştirmeye teşvik edecek bir neden yoktur çünkü diğer oyuncuların davranışlarını öngörebilmektedirler.

Önemli bir özelliği, sofistike dengeye ulaşabilmek için oyuncuların diğer oyuncuların düşünce süreçlerini ve rasyonel tepkilerini doğru bir şekilde tahmin etmeleridir. Bu tahminler, oyuncuların birbirlerinin davranışlarını takip etmeleri ve karşılıklı olarak en iyi stratejilerini belirlemeleriyle gerçekleşir. Baskın strateji yoksa, baskılanan stratejilerin sırayla elenmesiyle ulaşılan dengeyi ifade etmektedir. 1 ile 100 arasında bir rakam tutulduğu varsayımından yola çıkılarak, sürekli olarak en yüksek sayının 2/3'ü alınarak kişilerin tahmin ettiği sayı bulunmaya çalışılır. Bu yöntemin doğru olduğunu düşünen rasyonel

oyuncular olduğunu varsayalım. Sonuçta en son 1 tahmin edilen değer olur. Herkesin 1 tahmin ettiği sonucuna varılır. Bu da bize göstermektedir ki, gerçek dünyada böyle bir şeyin mümkün olması mümkün değildir. Yani kuramsal oyun teorisiyle davranışsal oyun teorisinin burada ayrıldığını görmekteyiz.

3.9. Tavuk Oyunu

Tavuk oyunu, oyun teorisi içinde yer alan bir strateji oyunudur. Bu oyun, iki oyuncunun aralarında risk alma ve çekinme dengesi üzerine kurulu bir rekabete girdikleri bir senaryoyu temsil eder (Wang ve ark., 2019; De Heus ve ark., 2010). Oyunda, oyuncuların birbirlerine karşı belirli bir eylemde bulunma veya kaçınma seçeneği vardır.

Bir tavuk oyunu genellikle iki oyuncunun birbirine doğru hareket ettiği bir oyun kurulumunu tanımlar. Oyuncular aynı yol üzerinde devam ederlerse, birbirlerine çarparlar; biri yoldan çekilirken diğeri çekilmezse, çeken "kaybeder" ve tavuk olarak etiketlenirken, ikinci oyuncu, dolaylı olarak daha cesur olan oyuncu, kazanır.

Tavuk oyununda, oyuncuların kararlarından dolayı ortaya çıkan 3 farklı sonuç vardır:

1. Eşit Risk Alma: Her iki oyuncu da risk alır ve ileri doğru ilerler. Bu durumda, çatışma yaşanır ve büyük bir tehlike ortaya çıkar.
2. Eşit Çekinme: Her iki oyuncu da çekinir ve yolun kenarına sapar. Bu durumda, çatışma önlendiği için bir kazadan kaçınılır, ancak her iki oyuncu da hedeflerine tam olarak ulaşamaz.
3. Korkak Tavuk: Bir oyuncu risk alırken, diğer oyuncu çekinir. Risk alan oyuncu ödül kazanırken, çekinen oyuncu geri kalır ve prestij kaybeder.

Tavuk oyunu için kazanç matrisi aşağıda gösterilmiştir.

		Sürücü 2	
		Yol değiştir	Yola devam et
Sürücü 1	Yol değiştir	0 , 0	-1 , +1
	Yola devam et	+1 , -1	-2 , -2

Burada baskın veya baskılanan strateji yok. Dolayısıyla Nash dengesini bulmamız gerekecek.

Sürücüler açısından en iyi cevaplara bakalım.

1. Sürücü

- Yola devam et EĞER ikinci sürücü Yol değiştirmişse
- Yol değiştir EĞER ikinci sürücü Yola devam ediyorsa

2. Sürücü

- Yola devam et EĞER birinci sürücü Yol değiştirmişse
- Yol değiştir EĞER birinci sürücü Yola devam ediyorsa

Görüldüğü üzere iki tane Nash dengesi vardır diyebiliriz. Bir sürücünün yol değiştirdiği ve diğer sürücünün yol değiştirdiği iki dengeden söz ediyoruz.

Tavuk oyunu, stratejik düşünme, risk alma ve karşı tarafın davranışını tahmin etme üzerine kuruludur. Oyuncular arasındaki dinamik, birbirlerinin stratejilerine tepki verme ve sonucu etkileme üzerine kuruludur. Oyun teorisi, tavuk oyunu gibi senaryolarda oyuncuların karar süreçlerini analiz ederek ve optimal stratejileri belirleyerek ilgilenir.

Problem: A firması ve B firması Türkiye'de yatırım yapmayı düşünmektedirler. Girecekleri sektörde piyasanın potansiyeli 20 milyar dolar olsun. Firmaların piyasaya girmek için yapacakları yatırım harcamalarının ise 14 milyar dolar olduğunu düşünelim. Tavuk oyunu'na göre kazanç matrisine bakalım.

		A Firması	
		Piyasaya Gir	Piyasaya Girme
B Firması	Piyasaya Gir	-4 , -4	+6 , 0
	Piyasaya Girme	0 , +6	0 , 0

Görüldüğü üzere, A firması ve B firması ikisi birden piyasaya girerse, potansiyel olarak kabul edilen 20 milyar doları bölüşeceklerdir. İki firmada 10'ar milyar dolar kazanacak ama giriş bedeli olarak 14 milyar dolarlık bir yatırım yaptıkları için 3'er milyar dolar zarar edeceklerdir. Sadece bir firmanın piyasaya girdiği iki denge bulunmaktadır. A firması tek başına piyasaya girerse potansiyelin tamamını ele geçirecek ve yaptığı yatırım maliyeti çıkartıldığında 6 milyar dolarlık bir kazanç sağlayacaktır. Aynı durum B firması içinde geçerlidir. İki firmanın piyasaya girmemesi durumunda ise herhangi bir kazanç ya da kayıp olmayacaktır.

Tespit edilen 2 Nash dengesi: Bir firmanın piyasaya girdiği diğerinin girmediği durumlar.

(Piyasaya Gir , Piyasaya Girme) ve (Piyasaya Girme , Piyasaya Gir)

Akla gelen soru şu olacaktır. Bu iki firmadan herhangi birisi kendi çıkarı doğrultusunda bu dengeyi bozabilir mi?

Bu durumda firmaların belirleyeceği stratejiler oyunu yapacakları hamlelere göre sıralı bir oyun haline getirebilir. Mesela B firması 6 milyar dolarlık bir yatırım yaparak o bölgeye bir fabrika açsın ve ürün tedarik süresini ve maliyetini azaltacak duruma gelsin. Bu durumda kazanç matrisimizi yeniden oluşturalım.

		A Firması	
		Piyasaya Gir	Piyasaya Girme
B Firması	Piyasaya Gir	-4 , -4	+6 , 0
	Piyasaya Girme	-6 , +6	-6 , 0

Bu senaryoda görüldüğü üzere bir tane baskın strateji görülmektedir. B firması yaptığı bu yatırımla dengeyi bozmuş, durumu kendi lehine çevirmiş ve A firmasının piyasaya girmesini engellemiştir.

Tespit edilen Nash dengesi: (Piyasaya Gir, Piyasaya Girme) olarak görülmektedir.

3.10. Koordinasyon Oyunları

3.10.1. Çiftlerin Savaşı – Cinsiyetler Savaşı

"Çiftlerin Savaşı" olarak da bilinen Oyun Teorisi'nde bir senaryo, iki oyuncu arasındaki çatışma veya etkileşimi modellemek için kullanılır. Bu senaryoda, oyuncuların her biri stratejilerini seçer ve bu seçimler sonucunda her bir oyuncunun kazançları veya kayıpları belirlenir.

"Çiftlerin Savaşı" adı, klasik bir senaryoya atıfta bulunur. Bu senaryoda, iki sevgili arasında bir çatışma vardır. Eğer her ikisi de birbirlerine sadık kalırsa, ilişki sürer ve her iki oyuncu da belli bir kazanç elde eder. Ancak, bir oyuncu sadakatsizlik yaparsa ve diğer oyuncu sadık kalırsa, sadakatsiz oyuncu daha fazla kazanç elde ederken, sadık oyuncu kaybeder. Eğer her ikisi de sadakatsizlik yaparsa, ilişki sona erer ve her iki oyuncu da bir kayıp yaşar.

Oyun teorisi, bu tür senaryolarda nasıl optimal stratejilerin belirleneceğini ve oyuncuların en iyi sonuçları elde etmek için nasıl hareket edeceğini araştırır. Bu çatışma senaryolarında, oyuncular genellikle

kazançlarını veya kayıplarını temsil eden bir "kazanç matrisi" kullanılır. Bu matris, oyuncuların her bir strateji kombinasyonu için elde edecekleri kazançları veya kayıpları gösterir.

Oyuncular, kendi kazançlarını veya kayıplarını maksimize etmek amacıyla stratejilerini seçerken, genellikle Nash dengesi gibi kavramlar kullanılır. Nash dengesi, her oyuncunun stratejilerini verildiğinde, hiçbir oyuncunun tek taraflı olarak stratejisini değiştirerek daha iyi bir sonuç elde edemeyeceği bir durumu ifade eder.

"Çiftlerin Savaşı" senaryosu, genel olarak, bir oyuncunun sadık kalmayı seçmesi durumunda diğer oyuncunun da sadık kalmayı seçmesinin daha iyi bir sonuç sağladığı bir Nash dengesine sahiptir. Ancak, bu senaryo bazı değişikliklere tabi tutulabilir ve farklı kazanç yapıları veya koşullar altında farklı sonuçlar ortaya çıkabilir.

Cinsiyetler savaşı oyununda, Pareto etkinliği gözlemlenir. Bu durum, oyuncuların strateji profillerine bakıldığında, hiçbir oyuncunun kendi faydasını azaltmadan diğer oyuncunun faydasını artıramayacağı anlamına gelir. Oyunda çatışmalar olsa da iş birliği yapmanın daha faydalı olduğu görülmektedir. Bu tür bir durum, ekonomik alanda birçok örnekte karşımıza çıkar. Örneğin, iki farklı firmanın faaliyet gösterdiği bir sektörde, her bir firmanın ürettiği ürünler için farklı standart tercihleri olabilir. Ancak, her iki firmanın da aynı standartlarda üretimin tüketicileri teşvik edeceğini bilerek birlikte hareket etmeleri, her iki firma için daha faydalı olacaktır.

Problem: Bir çift evde televizyon izleyerek vakit geçirmek istemektedirler. Kadın dizi film izlemek isterken erkek maç izlemek istemektedir. Tercihleri farklı olsa dahi, birbirlerini çok seven bu çift tercihlerini karşı taraf için değiştirebileceklerdir.

		Kadın	
		Maç	Dizi
Erkek	Maç	2 , 1	0 , 0
	Dizi	0 , 0	1 , 2

Bu oyunda, strateji kümesi (maç, maç), (maç, dizi), (dizi, maç), (dizi, dizi) olarak belirlenmiştir. İki oyuncunun birlikte karar vermek zorunda olduğu bu oyunda, baskın strateji Nash dengesi bulunmamaktadır. Ancak, oyunda iki farklı Nash dengesi mevcuttur. Birincisi, (maç, maç) stratejisi kombinasyonudur, diğeri ise (dizi, dizi) stratejisi kombinasyonudur. Kadın oyuncu, erkeğin maç seçtiğini düşünürse, kendi çıkarları doğrultusunda maç seçmek daha faydalı olacaktır. Benzer şekilde, erkek oyuncu da kadının dizi seçtiğini düşünürse, diziyi seçmesi kendisi için daha faydalı olacaktır.

3.10.2. Geyik Oyunu

Bu oyunda, oyuncular avcılardan oluşmaktadır ve ortak amaçları geyiği avlamaktır. Avcıların temel özelliği aç olmaları ve geyiğin onları doyuracak olmasıdır. Eğer avcılar birlikte hareket ederlerse, geyiği avlama şansları en yüksek olacak ve her iki avcı da maksimum faydayı elde edecektir. Ancak iş birliği olmadığında geyik kaçacaktır, çünkü geyiği yakalamak sadece iş birliğiyle mümkündür. Avcılar tek başına tavşan avına odaklandıklarında pusuyu terk ederlerse, tavşanı ya avlayacaklar ya da avlayamayacaklardır. Bir avcı tavşanı yakalarsa, fayda düzeyi düşük olacak, ancak bu durumda hem bireysel hem de kolektif fayda düzeyi düşük olacaktır. Ancak iş birliği olduğunda hem bireysel hem de kolektif fayda düzeyi optimal bir seviyede olacaktır. Sonuç olarak, avcılar için iki seçenek vardır. Avcılar ya kolektif olarak hareket edip yüksek fayda elde edecekler ya da bireysel fayda peşinde koşacaklardır.

		Avcı 1	
		Geyik	Tavşan
Avcı 2	Geyik	2 , 2	0 , 1
	Tavşan	1 , 0	1 , 1

Yukarıda kazanç matrisi verilen geyik oyununda oyunda, iki farklı Nash dengesi bulunmaktadır: (Geyik, Geyik) ve (Tavşan, Tavşan) stratejileri. Bir oyuncu, rakip oyuncunun geyik stratejisini oynayacağını düşünüyorsa, en iyi hamle geyik stratejisini oynamaktır. Aynı şekilde, bir oyuncu rakibinin geyik stratejisini devam ettireceğini düşünüyorsa, o da geyik stratejisini devam ettirecektir. Eğer bir oyuncu tavşan stratejisini oynarsa, rakibi de tavşan stratejisini oynamayı tercih edecektir. Çünkü bu durumda geyik stratejisini oynamak dezavantajlı olacaktır. Tek başına geyik avlayamayacak ve eli boş dönecektir. Sonuç olarak, oyuncuların aynı stratejileri oynaması rasyoneldir. Yani, bir oyuncu tavşan stratejisini oynadığında diğer oyuncunun da tavşan stratejisini oynaması, geyik stratejisini oynadığında ise diğer oyuncunun da geyik stratejisini oynaması hem bireysel hem de toplam fayda açısından daha avantajlı olacaktır.

3.11. Karma Stratejiler

Tahminlere dayalı oyunlarda, genellikle tek bir Nash dengesi bulmak mümkün değildir. Çünkü oyuncuların ne yapacakları konusunda belirsizlikler vardır. Bu belirsizliklerin yorumlanabilmesi için karma stratejiler geliştirilmiştir. Karma stratejiler, birden fazla stratejinin olduğu durumlarda ortaya çıkar. Bazı oyunlarda birden fazla denge noktası mevcut olabilir. Bu durumda karma stratejiler, oyuncular için en iyi karar stratejilerini sunar. Bu sayede oyuncular, rakipleri karşısında bazı hamlelerinde bir stratejiyi kullanırken, diğer hamlelerinde farklı bir strateji veya stratejileri uygulama

fırsatına sahip olurlar. Bu karma stratejiler, oyuncuların belirsizlikleri yönetmelerine ve en iyiyi elde etmelerine yardımcı olur.

A= nxm bir Matris olarak kabul edilsin. I. oyuncunun n tane ve II. oyuncunun m tane pür stratejisi olduğunu düşünelim.

I. oyuncunun karma stratejisi;

$$i = 1,2,3,4 \dots, n\ için\ x_i\ \geq 0\ ve\ \sum_{i=1}^{n} x_i = 1$$

olacak biçimdeki X = $\{x_1, x_2, x_3, x_4, \dots, x_n\}$ vektörüdür.

II. oyuncunun karma stratejisi;

$$j = 1,2,3,4 \dots, m\ için\ y_j\ \geq 0\ ve\ \sum_{j=1}^{m} y_j = 1$$

olacak biçimdeki Y = $\{y_1, y_2, y_3, y_4, \dots, y_m\}$ vektörüdür.

Şimdi karma strateji kümelerini gösterelim.

I. Oyuncunun karma strateji kümesi;

$$S_n = \{\, \mathrm{X} = (x_1, x_2, x_3, x_4, \dots, x_n) : \forall \mathrm{i} = 1,2, \dots, n\ için\ x_i\ \geq 0\ ve\ \sum_{i=1}^{n} x_i = 1\}$$

II. Oyuncunun karma strateji kümesi;

$$S_m = \{\, \mathrm{Y} = (y_1, y_2, y_3, y_4, \dots, y_m) : \forall \mathrm{j} = 1,2, \dots, m\ için\ y_j\ \geq 0\ ve\ \sum_{j=1}^{m} y_j = 1\}$$

Olarak gösterilebilir.

I. ve II. oyuncunun S_n ve S_m karma stratejilerinin kümeleri kompakt ve konvekstir.

Kompakt: Bir küme, herhangi bir elemanının sonsuz uzaklıkta olmadığı ve sınırlı olduğu anlamına gelir. Yani, her elemanıyla arasında minimum ve maksimum uzaklık değerleri bulunur. Karma stratejilerin kümeleri kompakt olduğunda, oyuncuların stratejilerinin sınırlı bir aralıkta kaldığı ve önceden belirlenmiş bir sayıda seçenek olduğu anlaşılır.

Konveks: Bir küme, herhangi iki noktasını birleştiren doğru parçasının da kümenin içinde olduğu anlamına gelir. Karma stratejilerin kümeleri konveks olduğunda, oyuncuların iki strateji arasındaki herhangi bir noktayı seçebildiği

ve bu seçimlerin tümü de kümeye ait olduğu anlaşılır. Başka bir deyişle, oyuncuların stratejileri arasında herhangi bir karışımı seçebilirler.

S_n I. oyuncunun karma stratejiler kümesi ve $x \in S_n$ olarak kabul edilsin. Bu demektir ki;

$$\|x\| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2} \leq x_1 + x_2 + x_3 + \cdots x_n = 1$$

Yukarıdaki denklemde S_n kümesinin sınırlı olduğunu bulmuş olduk. Çünkü $x \in S_n$ için $\|x\| \leq 1$ olduğu açıkça görülmektedir.

Peki S_n kümesi kapalı mıdır? Hem kapalı hem sınırlı ise S_n kümesi kompakttır diyebileceğiz.

k=1,2,3…… için $x^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}\right) \in S_n$ olsun. Elimizde artık $(x^{(k)})$ dizisi var.

$\lim_{k\to\infty} x^{(k)} = x^{(*)}$ olsun.

$x^{(*)} = \left(x_1^{(*)}, x_2^{(*)}, x_3^{(*)}, \dots, x_n^{(*)}\right) \in S_n$ olduğunu gösterelim.

$\lim_{k\to\infty} x^{(k)} = x^{(*)}$ olduğundan, i = 1,2,3,…,n için $x_i^{(k)} \to x_i^{(*)}$ olur.

$x^{(k)} \in S_n$ olduğundan, k=1,2,3…… için $x_i^{(k)} \geq 0$ olur.

$k \to \infty$ iken $x_i^{(k)} \to x_i^{(*)}$ olduğundan $x_i^{(*)} \geq 0$ olur.

$x^{(k)} \in S_n$ olduğundan, $\sum_{i=1}^{n} x_i^{(k)} = 1$ olur.

$\forall$ k=1,2,3…… için $k \to \infty$ iken $x_i^{(k)} \to x_i^{(*)}$ olduğundan,

$$\sum_{i=1}^{n} x_i^{(*)} = 1 \; olur.$$

Demektir ki, $x^{(*)} = \left(x_1^{(*)}, x_2^{(*)}, x_3^{(*)}, \dots, x_n^{(*)}\right) \in S_n$

Buda bize S_n kümesinin kapalı olduğunu göstermektedir. Dolayısıyla hem sınırlı hem de kapalı olduğunu gösterdiğimize göre, S_n kümesinin kompakt olduğunu ispatlamış oluyoruz. Şimdi konveks olduğunu ispatlayalım.

$x^{(1)} = \left(x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}\right), x^{(2)} = \left(x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(2)}\right) \in S_n$ ve

$\alpha \in [0,1]$ olsun. $\alpha x^{(1)} + (1-\alpha)x^{(2)} \in S_n$ olduğunu ispatlayalım.

$$\alpha x^{(1)} + (1-\alpha)x^{(2)} = \alpha\left(x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}\right) + (1-\alpha)\left(x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(2)}\right)$$

$$= \left(ax_1^{(1)}, \alpha x_2^{(1)}, \dots, \alpha x_n^{(1)}\right) + \left((1-\alpha)x_1^{(2)}, (1-\alpha)x_2^{(2)}, \dots, (1-\alpha)x_n^{(2)}\right)$$

$$= \left(\alpha x_1^{(1)} + (1-\alpha)x_1^{(2)}, \alpha x_2^{(1)} + (1-\alpha)x_2^{(2)}, \alpha x_n^{(1)} + (1-\alpha)x_n^{(2)}\right)$$

$x^{(1)}$ ve $x^{(2)} \in S_n$ olduğundan;

$x_i^{(1)} \geq 0$ ve $x_i^{(2)} \geq 0$ olur. $i = 1,2, \dots, n$ için;

$\alpha x_i^{(1)} + (1-\alpha)x_i^{(2)} \geq 0$ olur. (1)

$$\sum_{i=1}^{n} x_i^{(1)} = 1 \; ve \sum_{i=1}^{n} x_i^{(2)} = 1$$

$$\sum_{i=1}^{n} \left[\alpha x_i^{(1)} + (1-\alpha)x_i^{(2)}\right] = \alpha \sum_{i=1}^{n} x_i^{(1)} + (1-\alpha) \sum_{i=1}^{n} x_i^{(2)}$$

$= \alpha + (1-\alpha) = 1$ (2)

Denklem (1) ve Denklem (2)'ye göre;

$\alpha x^{(1)} + (1-\alpha)x^{(2)} \in S_n$ Denilebilir. Buda bize S_n kümesinin konveks olduğunun ispatıdır. Aynı durum S_m kümesi içinde geçerlidir.

Karma stratejiler için şunu söyleyebiliriz. I. oyuncu mevcut pür stratejilerden bir i stratejisini x_i olasılıkla seçecektir. II. oyuncu ise, j stratejisini y_j olasılıkla seçecektir. Eğer oyunda I. oyuncu i. stratejisini 1 olasılıkla ve diğer tüm pür stratejilerini 0 olasılıkla oynadığında sadece i. pür strateji ile oynadığı söylenebilir.

$X = \{0,0,0,0 \dots, 1.0, \dots, 0\}$

Aşağıdaki getiri matrisine göre;

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{11} & a_{11} & \dots & a_{2m} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}$$

I. oyuncu $X = \{x_1, x_2, x_3, \dots, x_n\}$karma stratejisi ve II. oyuncu II_j pür stratejisi seçtiğinde yani j. bileşeni 1 ve diğer bileşenleri 0 olan bir karma strateji, I. oyuncunun beklenen kazancı;

$$h(X, II_j) = \sum_{i=1}^{n} x_i a_{ij}, j = 1,2,3, \dots m$$

olur. Benzer şekilde, I. oyuncu I_i pür stratejisi ve II. oyuncu,

$Y = \{y_1, y_2, y_3, \dots, y_m\}$karma stratejisini seçerse I. oyuncunun beklenen kazancı;

$$h(I_i, Y) = \sum_{j=1}^{n} a_{ij} y_j, i = 1,2,3, \dots n$$

olur. Bundan dolayı;

I. oyuncu $X = \{x_1, x_2, x_3, \dots, x_n\}$,

II. oyuncu $Y = \{y_1, y_2, y_3, \dots, y_n\}$ karma stratejisini seçtiklerinde I. oyuncunun beklenen kazancı;

$$h(X, Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} x_i a_{ij} y_j$$

olarak bulunur. Matris olarak göstermek istersek;

$h(X, Y) = XAY^T$ olarak gösterilir. T ifadesi Transpoze demektir.

$Y^T = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$ bu matris $Y = \{y_1, y_2, y_3, \dots, y_m\} \in S_m$ vektörünün transpozesidir. Y^T için de vektördür deriz.

Şimdi Karma stratejilerin mantığını anlamamızı sağlayacak bir oyun üzerine konuşalım. Bu oyun Yazı-Tura oyunu dediğimiz, içerisinde iki olasılığı barındıran, birinin kaybederken diğerinin kazandığı eşleşen paralar olarak isimlendirilen oyun.

3.11.1. Eşleşen Paralar

Eşleşen paralar oyununda, oyuncular ellerindeki parayı havaya atarak yazı veya tura sonucunu birbirlerine göstermeyi seçerler. Eğer oyuncuların seçimleri aynı olursa, birinci oyuncu ikinci oyuncuya 1 birim öderken; eğer oyuncuların seçimleri farklı olursa, ikinci oyuncu birinci oyuncuya 1 birim öder. Oyuncular, kendi kazançlarını odak noktası olarak ele alarak bu durumu değerlendirirler. Aşağıdaki strateji matrisinde gösterildiği gibi, bu oyunda saf strateji Nash dengesi bulunmamaktadır. Çünkü bu oyunda durağan bir denge yoktur ve bir oyuncunun kazancı diğer oyuncunun kaybıyla sonuçlanır. Oyunculardan biri kazandığında diğeri kaybeder, bu da oyunda sıfır toplamlı bir yapıyı temsil etmektedir.

		Oyuncu 2	
		Yazı	Tura
Oyuncu 1	Yazı	Kaybet ,Kazan	Kazan , Kaybet
	Tura	Kazan , Kaybet	Kaybet , Kazan

Bu oyun sıfır toplamlı bir oyundur. Yani bir oyuncunun kazanması diğer oyuncunun kaybetmesiyle sonuçlanır. 1 birim kazanç diğer taraf için -1 birim kayıp anlamına gelmektedir. Buna göre oluşturulan kazanç matrisi aşağıda sunulmuştur.

		Oyuncu 2	
		Yazı	Tura
Oyuncu 1	Yazı	-1 , 1	1 , -1
	Tura	1 , -1	-1 , 1

Kazanç matrisine baktığımızda baskın bir stratejinin olmadığını görmekteyiz. Bu durumda en iyi cevaplara bakmalıyız. En iyi cevaplara baktığımızda ise, Oyuncu 1 eğer yazıyı seçerse oyuncu 2 kendi çıkarları doğrultusunda yazı derdi. Ya da oyuncu 2 yazıyı seçerse oyuncu 1 tura derdi. Diğer ihtimallerde de bir değişiklik olmayacaktır. Ortaya çıkan sonuç, en iyi cevaplarda herhangi bir kesişmenin olmadığıdır. Her iki oyuncuda saf strateji yani tek bir strateji kullandığında nash dengesi bulunmamaktadır ama John

Nash'e göre sonlu oyunlarda bir nash dengesi bulunmaktadır. İşte bu durumlarda saf stratejiler yerine karma stratejiler ön plana çıkmaktadır. Karma stratejiden kastedilen bir oyuncunun bir stratejiyi oynama ihtimali olabilir. Oyuncu 1'in yazı ya da tura seçme ihtimali için düşünecek olursak %50 ihtimalden bahsedebiliriz. İhtimalle beraber orandan da bahsedebiliriz. Oyuncu 1 100 defa bu oyunu oynarsa 50 defa yazı 50 defa tura oynayabilir. Oyuncu 1'in olduğu popülasyona 100 kişi varsa bunlardan 50'si yazı 50'si tura oynayabilir. Bu durumlar karma stratejilerle açıklanabilir. Bu oyun için karma stratejileri bulmaya çalışalım.

		Oyuncu 2		
		Yazı	Tura	
Oyuncu 1	Yazı	-1 , 1	1 , -1	(p)
	Tura	1 , -1	-1 , 1	(1-p)
		q	(1-q)	

Oyuncu 1 Yazı Seçerse → Beklenen Fayda = (-1)q + (1)(1-q) = 1-2q
Oyuncu 1 Tura Seçerse → Beklenen Fayda = (1)q + (-1)(1-q) = 2q-1
İki oyuncu içinde beklenen fayda birbirine eşittir.
1-2q = 2q-1 eşitliğinden q= 1/2 bulunur.
Oyuncu 2 Yazı Seçerse → Beklenen Fayda = (1)p + (-1)(1-p) = 2p-1
Oyuncu 2 Tura Seçerse → Beklenen Fayda = (-1)p + (1)(1-p) = 1-2p
İki oyuncu içinde beklenen fayda birbirine eşittir.
2p-1 = 1-2p eşitliğinden p= 1/2 bulunur.
Karma Nash Dengesi: { Oyuncu 1 (1/2,1/2) , Oyuncu 2 (1/2,1/2) }

Buda göstermektedir ki iki oyuncuda oyunu %50 kazanma ihtimaliyle oynamaktadırlar.

KAYNAKÇA

Abraham, I., Alvisi, L., & Halpern, J. Y. (2011). Distributed computing meets game theory: combining insights from two fields. Acm Sigact News, 42(2), 69-76.

Acar, V., & Ünal, S. (2022). Portföy Optimizasyonuna Yönelik Ampirik Bir Karşılaştırma: Oyun Teorisi ve Modern Portföy Teorisi. Balkan Sosyal Bilimler Dergisi, 11(21), 15-26.

Adler, N., Brudner, A., & Proost, S. (2021). A review of transport market modeling using game-theoretic principles. European Journal of Operational Research, 291(3), 808-829.

Askari, G., Gordji, M. E., & Park, C. (2019). The behavioral model and game theory. *Palgrave Communications*, *5*(1), 1-8.

Aydın, G., & Karabacak, H. (2023). Oyun Teorisi Perspektifinden Çatışma Yönetim Stratejilerinin Karşılıklı Etkileşimi ve Denetçilere Yönelik Bir Uygulama. Süleyman Demirel Üniversitesi Vizyoner Dergisi, 14(38), 607-625.

Ayres, R. U. (2020). Capitalism vs. Socialism: A Conflict of Ideas. On Capitalism and Inequality: Progress and Poverty Revisited, 77-84.

Bekmez, S. ve Çalış F. (2011). Oyun Teorisi Çerçevesinde Türk Bankacılık Sistemi ve Asimetrik Bilgi Problemi. Süleyman Demirel Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi, 16(2), ss.79- 96.

Bhaumik, A., Roy, S. K., & Weber, G. W. (2020). Hesitant interval-valued intuitionistic fuzzy-linguistic term set approach in Prisoners' dilemma game theory using TOPSIS: a case study on Human-trafficking. Central European Journal of Operations Research, 28, 797-816.

Bisht, M., & Dangwal, R. (2023). Fuzzy ranking approach to bi-matrix games with interval payoffs in marketing-management problem. International Game Theory Review, 25(01)

Camerer, C. F. (1991). Does strategy research need game theory?. Strategic Management Journal, 12(S2), 137-152.

Chen, Y., Zhao, J., Hu, J., Wan, S., & Huang, J. (2023). Distributed Task Offloading and Resource Purchasing in NOMA-enabled Mobile Edge Computing: Hierarchical Game Theoretical Approaches. ACM Transactions on Embedded Computing Systems.

Clemente, M., Fernández, F. R., & Puerto, J. (2011). Pareto-optimal security strategies in matrix games with fuzzy payoffs. Fuzzy Sets and Systems, 176(1), 36-45.

Colman, A. M. (2003). Cooperation, psychological game theory, and limitations of rationality in social interaction. Behavioral and brain sciences, 26(2), 139-153.

Conitzer, V., & Sandholm, T. (2008). New complexity results about Nash equilibria. Games and Economic Behavior, 63(2), 621-641.

Çiftçi, C. (2017). Jenerasyon Y'nin Yatırım Aracı Tercihleri: Oyun Teorisi Yaklaşımı. Karabük Üniversitesi Sosyal Bilimler Enstitüsü Dergisi, 7(2), 698-712.

De Heus, P., Hoogervorst, N., & Van Dijk, E. (2010). Framing prisoners and chickens: Valence effects in the prisoner's dilemma and the chicken game. Journal of Experimental Social Psychology, 46(5), 736-742.

Dekel, E., & Gul, F. (1997). Rationality and knowledge in game theory. Econometric Society Monographs, 26, 87-172.

Dey, S. (2018). A proof of work: Securing majority-attack in blockchain using machine learning and algorithmic game theory. International Journal of Wireless and Microwave Technologies, 8(5), 1-9.

Elgazzar, A. S. (2019). Unique solution to the quantum prisoner's dilemma game. Journal of the Physical Society of Japan, 88(3)

Elkind, E., & Leyton-Brown, K. (2010). Algorithmic game theory and artificial intelligence. AI Magazine, 31(4), 9-12.

Fang, F., Liu, S., Basak, A., Zhu, Q., Kiekintveld, C. D., & Kamhoua, C. A. (2021). Introduction to game theory. Game Theory and Machine Learning for Cyber Security, 21-46.

Fox, W. P. (2010). Teaching the applications of optimisation in game theory's zero sum and non-zero sum games. International Journal of Data Analysis Techniques and Strategies, 2(3), 258-284.

Fudenberg, D. ve Tirole J. (1991). Perfect bayesian equilibrium and sequential equilibrium. Journal of Economic Theory, 53, 236-260.

Genç, S. Y., & Kadah, H. (2018). Oyun teorisi ve Nash'in denge stratejisi. Iğdır Üniversitesi Sosyal Bilimler Dergisi, (14), 419-440.

Gunigari, H., & Chitra, S. (2023). Energy Efficient Networks Using Ant Colony Optimization with Game Theory Clustering. Intelligent Automation & Soft Computing, 35(3), 3557-3571.

Gupta, N., Soni, G., Mittal, S., Mukherjee, I., Ramtiyal, B., & Kumar, D. (2023). Evaluating Traceability Technology Adoption in Food Supply Chain: A Game Theoretic Approach. *Sustainability*, *15*(2), 898.

Guseinov, K. G., Akyar, E., & Düzce, S. A. (2010). Oyun teorisi: Çatışma ve anlaşmanın matematiksel modelleri [Game theory: Mathematical models of conflict and agreement]. Ankara: Seçkin Yayıncılık.

Hammerstein, P., & Selten, R. (1994). Game theory and evolutionary biology. Handbook of game theory with economic applications, 2, 929-993.

Hauert, C., & Szabó, G. (2005). Game theory and physics. *American Journal of Physics*, *73*(5), 405-414.

Hazra, T., & Anjaria, K. (2022). Applications of game theory in deep learning: a survey. Multimedia Tools and Applications, 81(6), 8963-8994.

He, Q., Cui, G., Zhang, X., Chen, F., Deng, S., Jin, H., ... & Yang, Y. (2019). A game-theoretical approach for user allocation in edge computing environment. IEEE Transactions on Parallel and Distributed Systems, 31(3), 515-529.

Hema, P., Paul, N. R., Čepová, L., Khan, B., Kumar, K., & Schindlerova, V. (2023). Complexity and Monitoring of Economic Operations Using a Game-Theoretic Model for Cloud Computing. Systems, 11(2), 50.

Holler, M. J. (1990). The unprofitability of mixed-strategy equilibria in two-person games: A second folk-theorem. Economics Letters, 32(4), 319-323.

Inegbedion, H., Asaleye, A., & Obadiaru, E. (2023). Competitive behaviour of major GSM firms' internet data pricing in Nigeria: A game theoretic model approach. *Heliyon*, e12886.

İzgi, B., & Özkaya, M. (2019). Matris normları ile bir matris oyununun adilliğinin gösterilmesi. International Journal of Advances in Engineering and Pure Sciences, 31(2), 126-132.

Jiang, Y., Kang, J., Niyato, D., Ge, X., Xiong, Z., Miao, C., & Shen, X. (2022). Reliable distributed computing for metaverse: A hierarchical game-theoretic approach. IEEE Transactions on Vehicular Technology, 72(1), 1084-1100.

Kealey, T., & Ricketts, M. (2014). Modelling science as a contribution good. Research Policy, 43(6), 1014-1024.

Kjeldsen, T. H. (2001). John von Neumann's conception of the minimax theorem: A journey through different mathematical contexts. Archive for history of exact sciences, 56(1), 39-68.

Köse, Y. (2014). Küresel finansal yaptırımlar: Oyun teorisi yaklaşımı ile ampirik bir uygulama. Maliye ve Finans Yazıları, 1(103), 10-20.

Kuhn, H. W., Harsanyi, J. C., Selten, R., Weibull, J., & Van Damme, E. (1996). The work of John Nash in game theory. Journal of Economic Theory, 69(1), 153-185.

McNamara, J. M. (2022). Game theory in biology: moving beyond functional accounts. The American Naturalist, 199(2), 179-193.

McNamara, J. M., & Leimar, O. (2020). Game theory in biology: concepts and frontiers. Oxford University Press, USA.

Moorthy, K. S. (1985). Using game theory to model competition. *Journal of Marketing Research*, *22*(3), 262-282.

Myerson, R. B. (1999). Nash Equilibrium and the History of Economic Theory. Journal of Economic Literature, s. 1067-1082.

Nash, J. (1951). Non-cooperative games. *Annals of mathematics*, 286-295.

Neshat, N., & Amin-Naseri, M. R. (2015). Cleaner power generation through market-driven generation expansion planning: an agent-based hybrid framework of game theory and particle swarm optimization. Journal of Cleaner Production, 105, 206-217.

Pereira, J. P. R. (2014). Use of a Game Theory model to simulate competition in Next Generation Networks. In *New Perspectives in Information Systems and Technologies, Volume 1* (pp. 387-397). Springer International Publishing.

Polat, M. (2021). Bankalar ve KOBİ'ler Arasındaki Kredi Sorunun Oyun Teorisi Çerçevesinde Çözümlenmesi. Cataloging-In-Publication Data, 355.

Razmi, P., Buygi, M. O., & Esmalifalak, M. (2020). A machine learning approach for collusion detection in electricity markets based on nash equilibrium theory. Journal of Modern Power Systems and Clean Energy, 9(1), 170-180.

Rençber, B. A. (2012). Karar vermede oyun teorisi tekniği ve bir uygulama. Uşak Üniversitesi Sosyal Bilimler Dergisi, 5(3), 96-107.

Rezek, I., Leslie, D. S., Reece, S., Roberts, S. J., Rogers, A., Dash, R. K., & Jennings, N. R. (2008). On similarities between inference in game theory and machine learning. Journal of Artificial Intelligence Research, 33, 259-283.

Samuelson, L. (2016). Game theory in economics and beyond. *Journal of Economic Perspectives*, *30*(4), 107-130.

Shi, Y., & Rong, Z. (2022). Analysis of Q-learning like algorithms through evolutionary game dynamics. IEEE Transactions on Circuits and Systems II: Express Briefs, 69(5), 2463-2467.

Shoham, Y. (2008). Computer science and game theory. Communications of the ACM, 51(8), 74-79.

Smith, J. M. (1979). Game theory and the evolution of behaviour. Proceedings of the Royal Society of London. Series B. Biological Sciences, 205(1161), 475-488.

Solomon, R. C. (1999). Game theory as a model for business and business ethics. Business Ethics Quarterly, 11-29.

Stolz, J. (2023). The theory of social games: outline of a general theory for the social sciences. Humanities and Social Sciences Communications, 10(1), 1-12.

Tanaka, T. (1994). Generalized quasiconvexities, cone saddle points, and minimax theorem for vector-valued functions. Journal of Optimization Theory and Applications, 81, 355-377.

Tengiz, M. (2020). Application of game Theory simulation in enterprise Management. In Colloquium-journal (No. 8 (60), pp. 136-140)

Uysal, F., Gülmez, M., & Çubukcu, H. A. (2017). Turizmde havayolu şirketlerinin fiyat belirleme politikaları ve oyun teorisi uygulaması. Uluslararası İktisadi ve İdari Bilimler Dergisi, 3(1), 5-19.

Von Neumann, J., & Morgenstern, O. (1947). Theory of games and economic behavior (2nd rev. ed.). Princeton University Press.

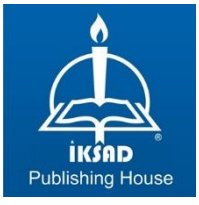
Wang, J., Zhengpeng, Y., Gillbanks, J., Sanders, T. M., & Zou, N. (2019). A power control algorithm based on chicken game theory in multi-hop networks. Symmetry, 11(5), 718.

Ye, M., Han, Q. L., Ding, L., & Xu, S. (2023). Distributed Nash equilibrium seeking in games with partial decision information: a survey. Proceedings of the IEEE, 111(2), 140-157.

Yu, H., Tseng, H. E., & Langari, R. (2018). A human-like game theory-based controller for automatic lane changing. Transportation Research Part C: Emerging Technologies, 88, 140-158.

Zeng, X. (2022). Game theory-based energy efficiency optimization model for the Internet of Things. Computer Communications, 183, 171-180.

Zhang, L., Wang, Y., Li, F., Hu, Y., & Au, M. H. (2019). A game-theoretic method based on Q-learning to invalidate criminal smart contracts. Information Sciences, 498, 144-153.



ISBN: 978-625-367-188-4