KSÜ Mühendislik Bilimleri Dergisi, 26(1), 2023
Araştırma Makalesi

KSU J Eng Sci, 26(1), 2023
Research Article

Kahramanmaras Sutcu Imam University
Journal of Engineering Sciences

Geliş Tarihi : 13.09.2022
Kabul Tarihi : 30.11.2022

Received Date : 13.09.2022
Accepted Date : 30.11.2022

2048 OYUNU İÇİN FARKLI ALGORİTMALARIN PERFORMANS ANALİZİ

PERFORMANCE ANALYSIS OF VARIOUS ALGORITHMS FOR THE 2048 GAME

Sabri AYLIK [1*](ORCID: 0000-0001-9204-6927)
Hüseyin HAKLI[2] (ORCID: 0000-0001-5019-071X)

[1,2] Necmettin Erbakan Üniversitesi, Bilgisayar Mühendisliği Bölümü, Konya, Türkiye

*Sorumlu Yazar / Corresponding Author: Sabri AYLIK, sabri.aylik@hotmail.com

ÖZET

2048, 4x4 ızgara üzerinde oynanan hücre kaydırmalı bir oyundur. 2048 oyunu kısa bir süre içerisinde insanlar arasında yayılarak önemli bir oynanma süresine sahip oldu. Popülerliğinin yanında, aksiyon sayısının az olmasına rağmen oluşabilecek çok fazla durumun bulunması nedeniyle yapay zeka araştırmacılarının dikkatini çekti. Tüm ihtimaller dikkate alındığında oyun tahtası üzerinde oluşabilecek durum sayısının 12^{16} olması ve oyundaki stokastik yapı dikkate alındığında oyunun zorluk derecesi görülmektedir. Literatürde 2048 oyunu üzerine uygulama içeren çalışmalar iki farklı yaklaşım altında değerlendirilmektedir. Bunlar öğrenme tabanlı yöntemler ve arama tabanlı yöntemlerdir. Bu çalışmada bu iki yöntem için en temel algoritmalar belirlendi. Arama tabanlı yöntemler olarak Derin Öncelikli Arama (Depth First Search - DFS) ve Monte Carlo Ağaç Arama (Monte Carlo Tree Search - MCTS) seçilirken, öğrenme tabanlı yöntem olarak Q-Öğrenme (Q-Learning) kullanıldı. Ayrıca algoritmalar budama ve rulet tekerliği gibi farklı teknikler ile uygulandı. Algoritmaların analizleri yapılırken skor ve maksimum karo metrikleri dikkate alındı. Skor metriği dikkate alındığında en başarılı algoritma DFS olurken, maksimum karo dikkate alındığında en başarılı yöntemin MCTS algoritması olduğu tespit edildi.

Anahtar Kelimeler: 2048 oyunu, pekiştirmeli öğrenme, derin öncelikli arama, Monte Carlo ağaç arama, algoritma analizi

ABSTRACT

2048 is a cell shift game played on a 4x4 grid. The 2048 game spread among people quickly and had a significant playing time. In addition to its popularity, it attracted the attention of artificial intelligence researchers since many situations can occur despite the small number of actions. The number of situations that can occur on the game board is 12^{16}; considering all the possibilities and the stochastic structure in the game are taken into account, the game's difficulty level is seen. In the literature, studies that include the implementation of the 2048 game can be evaluated mainly under two approaches. These are learning-based methods and search-based methods. In this study, basic algorithms for these two approaches have been determined. Q-Learning was used as a learning-based method when using Deep Priority Search (DFS) and Monte Carlo Tree Search (MCTS) as search-based methods. In addition, algorithms were applied with different techniques, such as pruning and the roulette wheel. Scores and maximum diamonds metrics were taken into account when analyzing algorithms. The most successful algorithm was DFS by score metric, while the MCTS algorithm was the most successful method under the maximum tile.

Keywords: 2048 game, reinforcement learning, depth-first search, Monte Carlo tree search, algorithm analysis

ToCite: AYLIK, S., & HAKLI, H., (2023). PERFORMANCE ANALYSIS OF VARIOUS ALGORITHMS FOR THE 2048 GAME. *Kahramanmaraş Sütçü İmam Üniversitesi Mühendislik Bilimleri Dergisi*, 26 (1), 67-77.

GİRİŞ

Günlük hayatta yaşamın nerdeyse her alanına girmiş olan yapay zekanın oyun sektörüne ilk adımları satranç oyunu üzerine oldu. 1950'lerden başlayarak birçok araştırmacının çalışmaları ve son yıllarda IBM'in vermiş olduğu destek sonucunda satranç alanında başarıya ulaşıldı (Newborn, 2000). Geliştirilen Deep Blue yapay zekası, dünya şampiyonu Garry Kasparov'u yapılan maçta yenmeyi başardı (Campbell et al., 2002). Deep Blue birçok bilgisayar bilimcisinin geliştirmiş olduğu arama algoritmalarını birleştirdi ve kazanmasının sebebi olarak ise bu durum gösterildi. Arama işlemi için en iyi oyuncuların eski oyun bilgileri kullanılarak bir veritabanı oluşturuldu. Oluşturulmuş olan bu veritabanında aramalar gerçekleştirerek en iyi hamle tespit edildi (Newborn, 2000). Yapay zeka araştırmacıları satranç alanında başarılı olduktan sonra daha zorlu bir oyun olan Go oyununa yöneldi. Go oyununun zor olmasının sebebi ise 10^{700} adet oyun durumu içermesiydi (Wang et al., 2016). Bu zorluk yapay zeka araştırmacılarının dikkatini çekmeyi başardı ve oyun üzerinde çalışmalar artış gösterdi. Yapılan çalışmalar sonucunda Deep Mind şirketinin 2016 yılında geliştirdiği AlphaGo programı 18 kez dünya Go şampiyonu olan Lee Sedol'u 5 maçlık seride 4:1 skor ile yenmeyi başardı (Siau and Wang, 2018; Wang et al., 2016). Lee Sedol'u yenmeyi başaran AlphaGo, uzman insanların oyunlarından danışmanlı öğrenme ve kendi kendine oynanan oyunlardan pekiştirmeli öğrenme tarafından eğitilen derin sinir ağlarının Monte Carlo Ağaç Arama yöntemi ile birleştirilmesi sonucunda oluşturulan bir bilgisayardı (Yu, 2016).

Araştırmacılar klasik oyunlarda başarı elde ettikten sonra bilgisayar oyunları üzerinde çalışmalar gerçekleştirmeye başladılar. Bu oyunlardan biri de 2048 oyunu oldu. 2048 oyunu, Gabriele Cirulli tarafından 2014 yılında bir Github projesi olarak geliştirildi. Oyunun yapımcısına göre yayınlandığı ilk haftada 2048 oyunu, 500 yıldan fazla oynama süresine sahipti (Jaśkowski, 2018). Devam eden iki hafta sonunda ise oyunun oynanma süresi 3000 yılı (Yeh et al., 2017). Bu kadar çok oynanmasının sebebi insanlar üzerinde bağımlılık oluşturması olarak tespit edildi. Oyunun kazanma kurallarının basit olmasına rağmen oyunu kazanabilmenin zor olması bu bağımlığının önemli bir sebebi olarak görüldü. Oyun sadece zaman geçirmek isteyen insanların ilgisini çekmedi. Aynı zamanda yapay zeka araştırmacılarının da ilgisini çekmeği başardı ve yapay zeka ile ilgilenen insanların test ortamı olarak kullanılmaya başlandı (Matsuzaki, 2017).

2048 oyunu için kullanılabilecek yöntemler hakkında yapay zeka araştırmacıları arasında ortak bir karar alınamadı. Bunun sebebi araştırmacıların bir kısmı bunun bir öğrenme problemi (pekiştirmeli öğrenme, yapay sinir ağları, N-Tuple, vb) olduğunu savunmakta (Kondo and Matsuzaki, 2019; Boris and Goran, 2017) iken diğer araştırmacılar ise bunun bir arama problemi (Minimax, Expectimax, Averaged Depth Limited Search, Breath First Search, vb) olduğunu öne sürdüler (Nie et al., n.d.; Rodgers and Levine, n.d.). Bu alanda son çalışmalarda 2048 oyununun daha çok öğrenme problemi olarak değerlendirildiği görüldü. 2022 yılında Weikai ve Kiminori oyunu bir öğrenme problemi olarak değerlendirerek Derin Sinir Ağları (Deep Neural Networks - DNN) yöntemini kullandılar. DNN yönteminin kullanma sebebi olarak ise AlphaStar, AlphaGo, DeepStack vb. bilgisayar oyunlarındaki başarısını ve gelişimini belirttiler (Weikai and Kiminori, n.d.). Sauren Ağustos 2022'de yayınlanan tez çalışmasında 2048 oyunu için iyileştirilmiş bir pekiştirmeli öğrenme yöntemi sundu. Önerilen sistemde pekiştirmeli öğrenmenin güvenliği garanti etmek için kullanılan koruma (shielding) yönteminden faydalandı ve açgözlü/stratejisiz ajanlara göre performansı belirgin şekilde iyileştirdi (Sauren K, Jansen N, and Strüber D, 2022). Bu konuda farklı çalışmalar bulunmasına ve artış göstermesine rağmen yayına dönüşmesi konusunda aynı artış görülememiştir. Bu makalede ise temel algoritmaların performans durumlarını görebilmek için hem öğrenme tabanlı hem de arama tabanlı yöntemler kullanıldı. Öğrenme tabanlı yöntem olarak Q-öğrenme yöntemi belirlenirken, arama tabanlı yöntemlerden DFS ve MCTS algoritmaları kullanıldı.

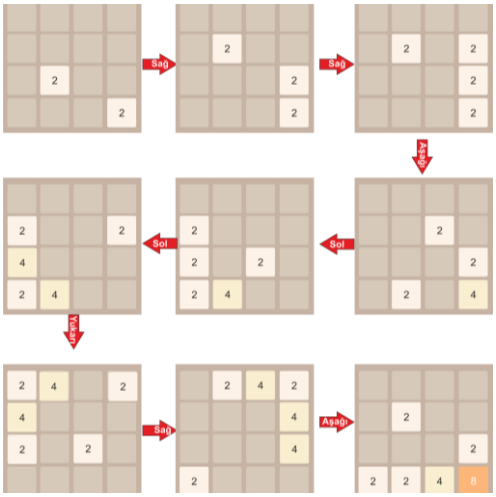
Bu çalışmada 2048 oyunu için DFS, MCTS ve Q-öğrenme algoritmalarının uygulanması ve sonuçlarının analizi yapıldı. Çalışma şu bölümlere ayrılmaktadır: Bölüm 2'de Materyal ve Metot, Bölüm 3'te 2048 Oyunu için uygulamalar ve Bölüm 4'te Bulgular ve Tartışma olarak sunuldu.

MATERYAL VE METOT

2048 Oyunu

2048 oyunu 4x4 ızgara üzerinde oynanan bir oyundur. Izgara üzerindeki her hücre ya boş ya da 2'nin bir üs kuvveti (2, 4, 8, 16 vb.) değerine sahip olmalıdır. Oyun farklı iki hücreye rastgele şekilde 2 veya 4 değerlerinden biri atanarak başlamaktadır. Kullanıcı dört farklı yönden (yukarı, aşağı, sol, sağ) birini seçmektedir. Sonrasında boş olmayan tüm hücreler seçilen yönde köşeye veya diğer boş olmayan hücrelerden birine çarpıncaya kadar kayar. Seçilen yön

doğrultusunda aynı değere sahip olan iki hücre çarpışırsa bu hücrelerin değerleri toplanır ve köşeye daha yakın olan hücreye atanır. Bu işlem oyun tahtası üzerinde boş hücre kalmayıncaya kadar devam eder. Oyunun amacı hücreleri birleştirerek en yüksek hücre değerine sahip olmaktır. Şekil 1'de oyunun yapısı ve durumlar arası geçişleri gösteren bir görsel sunulmaktadır.



Şekil 1. Oyunun Yapısı

Oyun içerisinde temel metrik maksimum karo değeridir. Ayrıca maksimum karo değeri dışında oyunda yapılan geçişlerden elde edilen 'skor' adında bir metrik daha vardır. Bu skor değeri Algoritma 1'de verilen algoritma ile hesaplanmaktadır (Cirulli, 2014).

Algoritma 1. Oyun tahtasında skor değerinin hesaplanması
Girdi: *değerler*
Çıktı: Skor

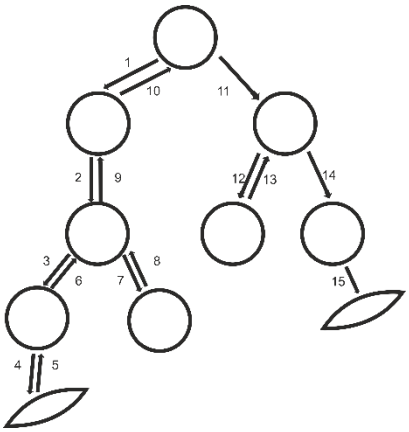
```
1  for satır=1 to 4 do:
2          for sutun 1 to 4 do:
3                  derece = log(degerler[satır][sutun],2)
4                  toplam_skor += (degerler [satır][ sutun]*(derece-1))
5          endfor
6  endfor
```

Skor ve maksimum karo arasında bir orantı bulunmasına rağmen değişim miktarı aynı seviyede ilerlememektedir. Örneğin iki farklı oyun tahtası durumunu ele alalım: Bunlardan birinde sadece bir adet 128 değeri bulunsun, diğerinde ise dört karo 64 değerine sahip olsun. Bu iki durum için ilk durumda maksimum karo değeri 128 olacaktır. İkinci durum için maksimum karo değeri 64 olacaktır. Fakat ilk durumun skor değeri 768 olurken ikinci durumun skor değeri 1280 olacaktır. Ayrıca maksimum karo metriğinde eşitlik durumu çok sıklıkla gerçekleşirken, skor metrikleri genellikle farklı olmakta ve durumlar arası daha net değerlendirme yapabilmek adına skor metriği avantaj sağlamaktadır. Bu sebeple performansların analizi için maksimum karonun yanında skor metriği de kullanılmıştır.

Derin Öncelikli Arama (Depth First Search)

Derinlik Öncelikli Arama (Depth First Search) bir ağaç tabanlı arama algoritmasıdır. Oluşabilecek tüm durumların tespit edilmesi ve en iyi durumun seçilmesi işleminde kullanılır. Derinlik tabanlı olarak çalışan bu algoritmada derinlik değeri arttıkça elde edilecek sonuçlarda doğrusal olarak artmaktadır. Ama derinlik değerinin artması işlem yükünü de arttırmaktadır. Bu sebeple zamansal olarak olumsuz etkilemektedir. Yukarıda belirtilenler dikkate alındığında derinlik değerinin seçilmesinde bu iki metrik dikkate alınmalıdır.

DFS yöntemi Şekil 2'de görüldüğü üzere önce bir durum için belirtilen derinliğe iner ve sonra diğer duruma geçer. Bu şekilde her durum için arama işlemini gerçekleştirmiş olur.



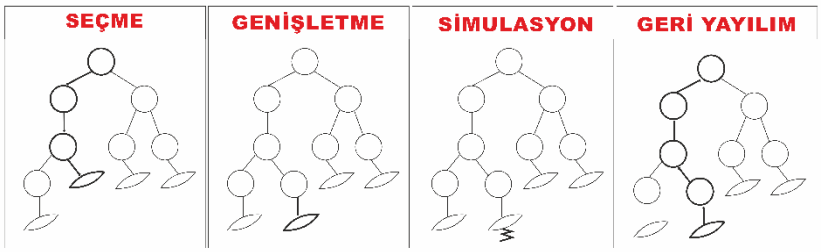
Şekil 2. DFS Çalışma Prensibi

Monte Carlo Ağaç Arama (Monte Carlo Tree Search)

Monte Carlo yöntemi, sayısal algoritmalar içinde uzun bir geçmişe sahiptir ve ayrıca çeşitli yapay zeka oyun oynama algoritmalarında, özellikle Scrabble ve Bridge gibi kusurlu bilgi oyunlarında önemli başarılara sahiptir. Monte Carlo Ağaç Arama (Monte Carlo Tree Search) algoritması yapılabilecek eylemler sonucunda oluşabilecek durumları simüle ederek karar uzayını tespit ettikten sonra bu uzay üzerinde rastgele örnekler alır ve sonuçlara göre bir arama ağacı oluşturarak belirli bir alanda optimal kararları bulmayı amaçlamaktadır (Browne et al., 2012). Şekil 3, MCTS yönteminin çalışma prensibini göstermektedir.

Algoritma önceden tanımlı bir hesaplama maliyetine (derinlik, zaman, bellek) ulaşıncaya kadar yinelemeli olarak bir arama ağacı oluşturur. Bu noktada arama işlemi duraklatılır. En iyi performansı gösteren eylem tespit edilir. Yukarıda anlatılan işlemler algoritmanın 4 farklı aşamasında gerçekleşir. Şekil 3'te görsel olarak verilen algoritmanın aşamaları şu şekilde anlatılabilir:

1. Seçim: Kök düğümden başlayarak en uygun düğüme ulaşılana kadar ağaçtan aşağı inmek için bir çocuk seçme politikası belirlenir.
2. Genişletme: Mevcut eylemlere göre ağacı genişletmek için bir (veya daha fazla) alt düğüm okunur.
3. Simülasyon: Yeni düğümlerden bir simülasyon, bir sonuca ulaşmak için varsayılan politikaya göre çalıştırılır.
4. Geri yayılım: Simülasyon sonucu, istatistiklerini güncellemek için seçilen düğüm aracılığıyla "yedeklenir" (yani geri yayılır).



Şekil 3. MCTS Çalışma Prensibi

Q-öğrenme (Q-Learning)

Pekiştirmeli öğrenme hayvan psikologlarının makine öğrenimi alanına kazandırdıkları bir yöntemdir (Mehta, 2014). Pekiştirmeli öğrenmede diğer makine öğrenimi yöntemlerindeki gibi girdi ve çıktı çifti istenmemektedir. Model kendi tecrübeleri ile öğrenme işlemini gerçekleştirir. En iyi sonuçları bulmak için yaptığı hamlelerden ödül ve cezalar elde eder. Q-öğrenme yaşam döngüsü Şekil 4'te verilmektedir.

Şekil 4. Q-öğrenme Yaşam Döngüsü

Q-öğrenme yönteminde yapılacak hamlenin seçimi aşamasında Denklem (1)'de belirtilen Belman denklemi kullanılmıştır. Bu denklemden modelin kendi kendine öğrenmesi aşamasında eylemlerin seçilmesi için yararlanılmaktadır. Oluşabilecek durumlar arasında analiz yapılır ve maksimum değeri veren eylem seçilerek işlem tekrarlanır.

$$V(s) = \max_a (R(s,a) + \gamma V(s')) \quad (1)$$

Denklem (1)'de *V(s)* belirli bir *s* durumunun değerini ifade ederken, *R(s,a)* ise *s* durumunda *a* eylemi gerçekleştirildikten sonra alınan ödülü temsil etmektedir. *s'* sonraki durumu ifade ederken, *V(s')* ise *s* durumunda *a* eylemi gerçekleştirildikten sonra oluşacak durumun değerini göstermektedir. Azaltma katsayısı *γ* ile durum geçişinden elde edilen ödüllerin etkisi kontrol edilmektedir.

Q-öğrenme, en temel pekiştirmeli öğrenme yöntemlerinden biridir. Pekiştirmeli öğrenme bir ajanın değişen çevre koşullarında nasıl davranacağını öğrendiği önemli bir tekniktir. Markov karar süreci pekiştirmeli öğrenmede ajanın değişen çevreye karşı yaptığı davranışların ödül ceza sistemi ile modellenmesidir.

Q-öğrenme mevcut verilerden mümkün olan en yüksek faydayı sağlayacak şekilde politikalar çıkarmayı amaçlamaktadır. Q-öğrenme öğrenme işlemi gerçekleştirmek için matematiksel bir arka plan kullanır. Ajan çevre ile etkileşimde bulunarak deneyim toplamakta ve sonrasında eldeki veriler ile politikayı geliştirmeyi amaçlamaktadır (Levine et al., 2020).

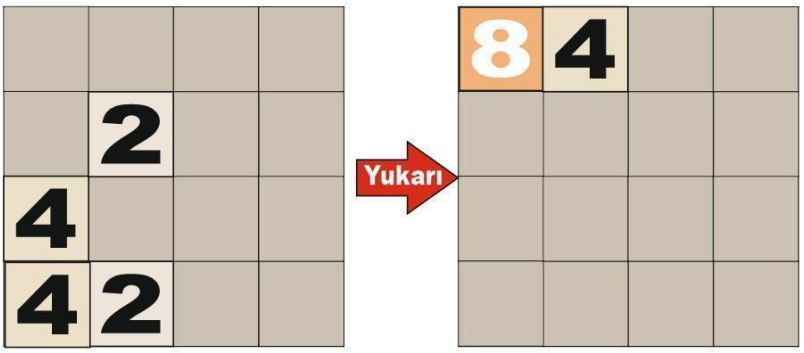
2048 OYUNU İÇİN ALGORİTMALARIN UYGULANMASI

Derin Öncelikli Arama (Depth First Search) Algoritmasının Uygulanması

DFS algoritması ağaç tabanlı bir yöntemdir. Bu yöntemde kök ve yapraklar bulunmaktadır. 2048 oyunu dikkate alındığında mevcut durum kökü oluştururken hamle sonrasında oluşabilecek durumlar yaprakları temsil etmektedir.

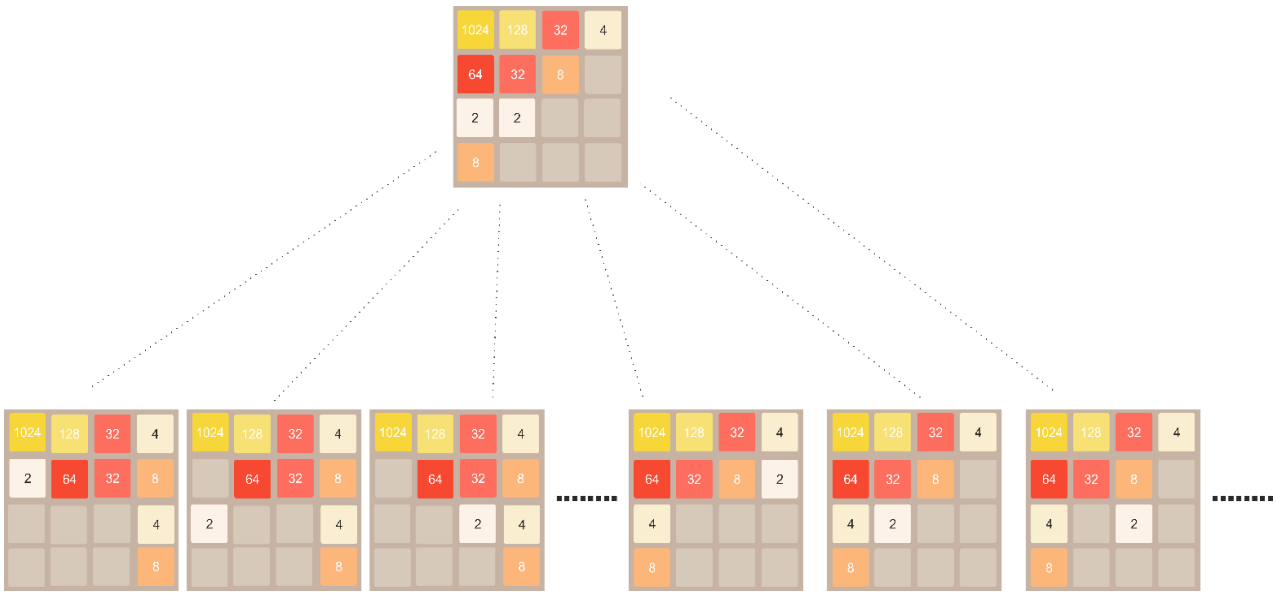
Arama ağacı oluşturulurken oyunda yapılabilecek dört hamle bulunmasına rağmen ağaç üzerinde oluşacak yaprak sayısı oyundaki stokastik durumdan dolayı bundan çok daha fazla olmaktadır. Bu durum Şekil 6 üzerinde açık bir şekilde görülebilmektedir. Ağaç üzerindeki bir kökten oluşacak yaprak sayısı hamle sonrasında oyun tahtası üzerindeki boş karo sayısı kadardır. Şekil 5 üzerinde de gösterildiği üzere Şekil 5.a'nın kök olduğu varsayarsak yapılan hamle sonrasında oluşan oyun tahtasında 14 adet boş karo olduğundan mevcut kökten oluşacak 14 yaprak bulunmaktadır. DFS algoritmasının diğer bir parametresi derinlik değeridir. Bu değer ağacın kaç hamle sonrasına kadar arama yapacağını belirler. Örneğin derinlik değeri 3 olan bir arama işleminde DFS algoritması 3 hamle boyunca oluşabilecek tüm durumları ağaç üzerinde oluşturur. Belirtilen derinlik değerine ulaşınca arama işlemini durdurur. Seçim işlemini gerçekleştirir.

Seçim işlemi yapılırken en optimal seçimi elde edilecek skor değerine göre yapmaktadır. Ağaç üzerindeki yapraklar (oluşabilecek durumlar) arasında en optimal durum belirlenir. Seçilen yaprağın sahip olduğu yön hamle yönü olarak seçilmiş olmaktadır. Şekil 6'da DFS yönteminde bir derinlikli ağaç yapısındaki muhtemel yapraklar gösterilmektedir.



(a) (b)

Şekil 5. Boş Karo Sayısının Tespiti



Şekil 6. DFS Üzerinde Yaprakların Oluşması

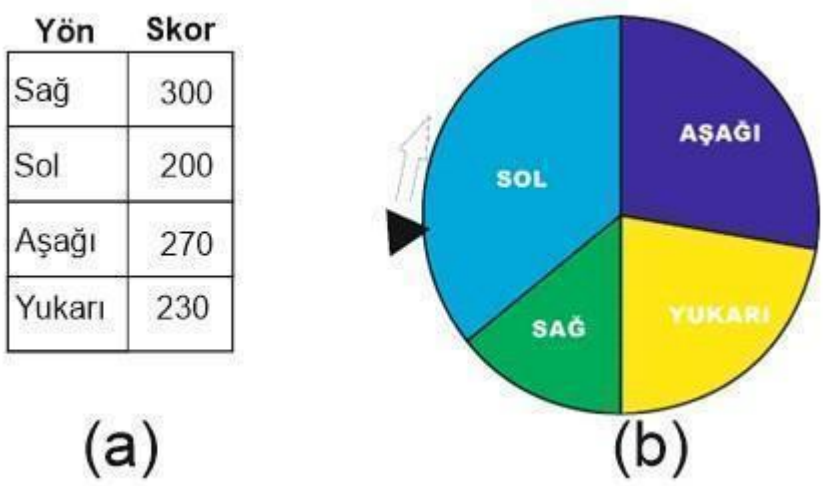
Monte Carlo Ağaç Arama (Monte Carlo Tree Search) Algoritmasının Uygulanması

MCTS temelde DFS algoritmasına benzer bir şekilde çalışmaktadır. MCTS algoritmasının DFS algoritmasından farkı ağaç yapraklarının genişletilmesi kısmında gerçekleşmektedir. Bu algoritma tıpkı DFS gibi bir arama ağacı oluşturmaktadır. Bu ağacın farkı derinlik değerindedir. DFS üzerindeki her yaprağın derinlik değeri eşit derecede iken MCTS algoritmasında yaprakların derinlik değerleri farklı olabilmektedir.

MCTS algoritmasında ilk adım arama ağacı üzerinde bir yaprak düğümü seçmektir. Başlangıçta her düğüm için sıfır değeri atanmaktadır. Yaprak düğümü seçilirken düğümlere atanmış olan bu değer dikkate alınmaktadır. Seçim yapıldıktan sonra seçilen düğüm kök kabul edilerek, bu kökün yaprakları oluşturulur. Bu sayede arama ağacı genişletilmiş olmaktadır. Sonrasında seçilmiş olan yaprak düğümünün skor değeri baz alınarak kök düğümden seçilen yaprak düğümüne izlenen yoldaki tüm düğümlere bir değer atanmaktadır. Bu değer düğümün bulunduğu derinlik ile ters orantılı olarak paylaştırılmaktadır.

Oyunun stokastik durumunu elimine etmek için MCTS ile birlikte rulet tekerleği yöntemi de kullanılmıştır. Rulet tekerleği karar uzayındaki sınıfların bulunma olasılıkları ile seçilme olasılıklarını doğru orantılı olduğunu varsayarak seçim gerçekleştiren bir algoritmadır. Rulet tekerleği yöntemi genellikle genetik ve evrimsel algoritmalarda kullanılır. Bunun sebebi mevcut arama yöntemlerinde N bireyden birini seçme işlemi O(N) karmaşıklıkta yapılmaktadır. Fakat bu işlem yerine rulet tekerleği kullanılarak O(1) karmaşıklık ile gerçekleştirilmektedir

(Lipowski and Lipowska, 2011). Rulet tekerleği yöntemi ile başarılı oyunlarda en çok hangi eylemlerin kullanıldığını dikkate alınmakta ve bu eylemlerin seçilme olasılığı arttırılarak MCTS ile beraber kullanıldığı çalışmalar bulunmaktadır (Jacobsen, Greve, and Togelius, 2014; Tong, n.d.). Rulet tekerleği yönteminin uygulanması Şekil 7'de gösterilmektedir. Şekil 7.a' da yapılan hamlelere karşılık elde edilecek skor değerleri mevcuttur. Şekil 7.b'de bu skor değerlerine ait oluşturulan rulet tekerleği bulunmaktadır. Şekil 7.a'da değer ne kadar yüksek ise o hamlenin şekil 7.b'de seçilme olasılığı o kadar yüksek olmaktadır. Hamle seçimi için tüm yönlere ait skor değerleri toplanır. Ardından sıfır ile toplam değer arasında rastgele bir sayı üretilir. Bu sayı hangi yönün bulunduğu aralığa düşerse o yön hamle yönü olarak seçilmektedir.



Şekil 7. Rulet Tekerleği Yönteminin Uygulanması

Q-öğrenme (Q-Learning) Algoritmasının Uygulanması

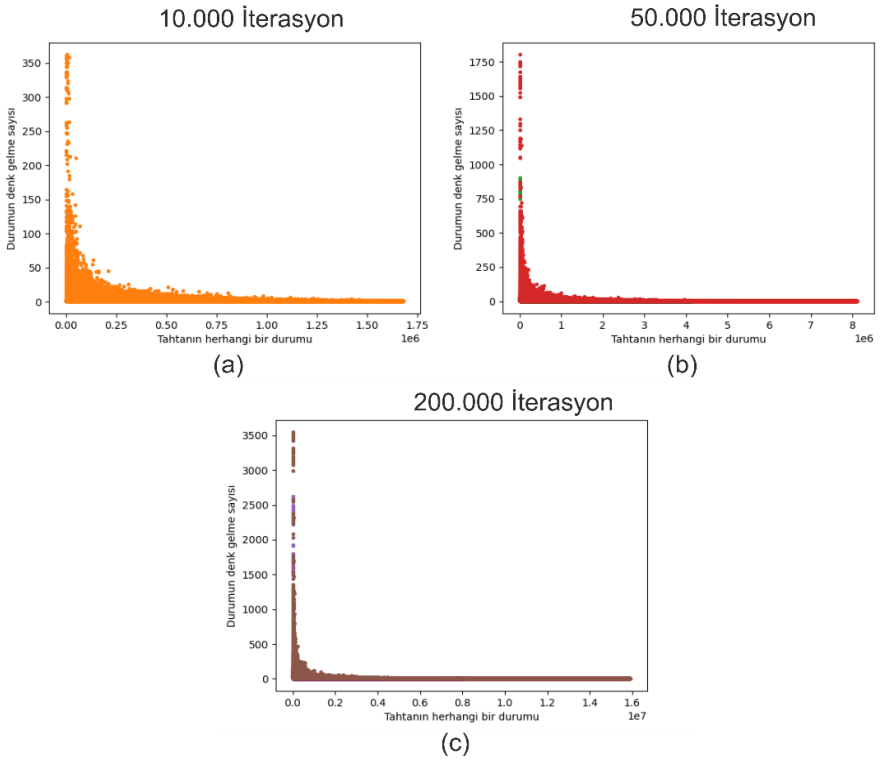
Q-öğrenme modelinin geliştirilmesi için durumlar ve eylemler arasında bir ödül sisteminin bulunması gerekmektedir. 2048 Oyunu için bu sistem eylem sonucunda oyun tahtasından elde edilecek skora göre yapılmaktadır. 2048 oyunu kendi kendine oynatılmaya başlanmaktadır. Yapılacak eylem seçilirken eylem sonrasında oluşan durumlar dikkate alınarak dört farklı yönden biri seçilmektedir. Bu işlem yapılabilecek hamle kalmayıncaya kadar devam etmektedir. Oyun sonlandıktan sonra elde edilen skor değeri daha önce belirlenmiş olan γ ile çarpılarak azaltılmaktadır. Bu değerin amacı oyun boyunca yapılan tüm hamleler arasından ilk hamlelerin daha yüksek ödül almasını sağlamaktadır. Böylece ilk hamleden son hamleye doğru azaltılarak ödüller verilmektedir. Verilen ödül değeri başlangıç hamleleri için daha yüksek olurken sonraki hamleler için daha küçük olmaktadır.

Oluşturulan ödül sistemi kayıt altına alınarak öğrenme işlemi gerçekleştirilmektedir. Bu sayede herhangi bir durum için farklı ödül değerleri verilerek en optimal ödül değerine ulaşması amaçlanmaktadır. Bu sebeple oyun $2x10^5$ iterasyon ile eğitildi. Eğitim işlemi tamamlandıktan sonra oyun kaydedilmiş veriler doğrultusunda kararlar almaktadır. Böylece herhangi bir durum için hamle seçimi yapılırken en yüksek ödül değerine sahip olan yön hareket yönü olarak seçilmektedir.

BULGULAR VE TARTIŞMA

Q-öğrenme yönteminin eğitilmesi aşamasında farklı öğrenme ve azaltma parametre değerlerinin kullanılmasına ve arttırılan iterasyon sayısına rağmen istenen başarıya ulaşamadığı görülmüştür. Bunun sebebi oyun tahtası üzerinde keşfedilecek çok fazla durumun olmasıdır. Öğrenme sürecinde oynatılan $2x10^5$ oyuna rağmen model kayda değer bir öğrenme gerçekleştirememiştir.

Oyun tahtası üzerinde oluşabilecek 12^{16} farklı durum bulunmaktadır. Oynatılan $2x10^5$ oyunda sadece 16×10^6 durum tespit edilebilmiştir. Eldeki veriler tespit edilebilen durumların sayısının ne kadar az olduğunu göstermektedir. Bu sebepten dolayı model öğrenmeyi başaramamıştır. Şekil 8 oynatılan oyun sayısına karşılık oyun tahtası üzerindeki durumların denk gelme sayılarını göstermektedir. İterasyon sayısının arttırılmasının keşfedilmemiş durumlara yeterli miktarda etkisi bulunmamıştır.

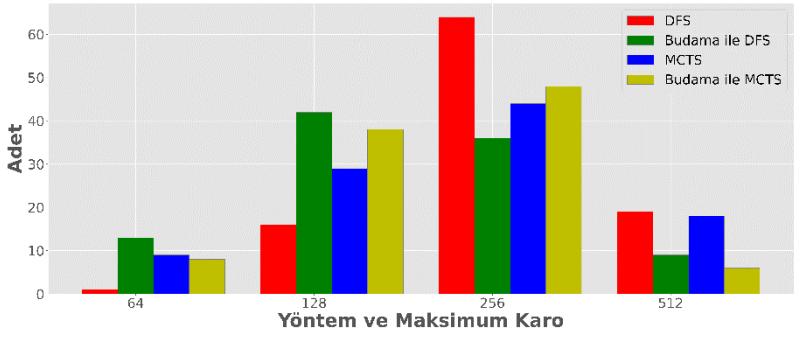


Şekil 8. Durumlar ve Denk Gelme Sayıları

Ağaç tabanlı modellerde herhangi bir eğitim durumu olmadığından geliştirilen modelden doğrudan sonuçlar alınabilmektedir. Ağaç tabanlı arama yöntemlerinin sonuçlarının karşılaştırılabilmesi için tüm yöntemlerde yer alan derinlik parametresinin belirlenmesi gerekmektedir. Derinlik değeri, işlem yükünün durumu da değerlendirilerek 3 olarak belirlenmiştir. Derinlik değeri belirlendikten sonra her yöntem 100 kez çalıştırıldı. Elde edilen sonuçlar aşağıda belirtilen metrikler dikkate alınarak karşılaştırılmıştır:

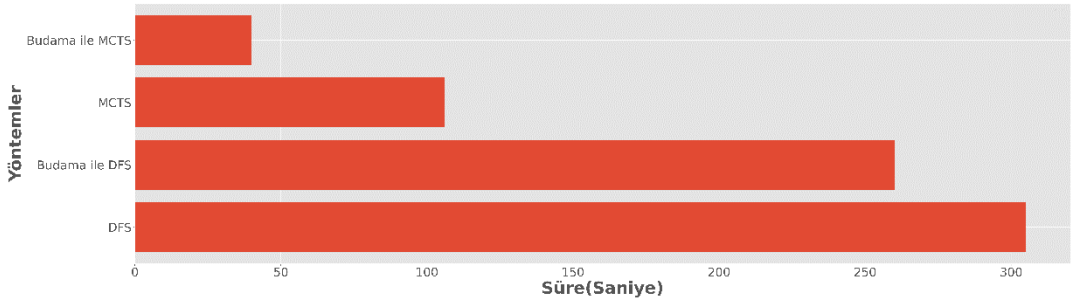
- Maksimum karo
- Zaman
- Skor
- Ortalama Hamle

Şekil 9'de maksimum karo değerine ait sonuçlar gösterilmiştir. Grafiğin yatay ekseni ulaşılabilecek maksimum karoları, dikey ekseni yatay eksendeki değere ulaşma adedini, yatay eksen üzerindeki farklı renkler ise yöntemleri ifade etmektedir. Şekil 9 incelendiğinde, elde edilen maksimum karo değeri 512 için en iyi yöntemlerin DFS ve MCTS olduğu görülmektedir. Bu yöntemlere uygulanan budama işlemlerinin başarı performansını olumsuz yönde etkilediğini gözlenmektedir. Yapılan işlemin olumsuz bir sonucu olmasına rağmen kullanılmasının sebebi iş yükünü azaltmasıdır. Böylece Şekil 10'da verilen grafikte görüldüğü üzere zamandan kazanç sağlamaktadır.



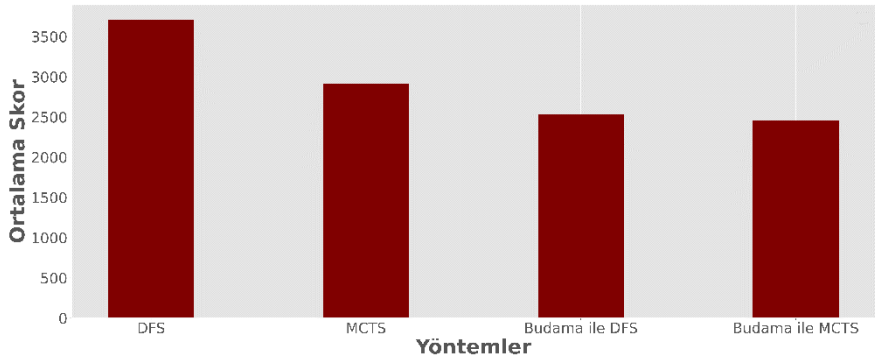
Şekil 9. Maksimum Karo Adet Grafiği

Şekil 10, yöntemlerin oynatılan 100 oyun için kaydedilen oyun sürelerinin ortalamasını göstermektedir. Bu doğrultuda Şekil 9'de en iyi başarıyı elde etmiş olan DFS yöntemi zaman göz önüne alındığında en kötü sonucu veren yöntem olduğu görülmektedir. Buna karşın Şekil 9'de en kötü başarıyı veren Budama ile MCTS ise oyun süresi dikkate alındığında diğer durumlara göre daha ön plana çıkmaktadır.



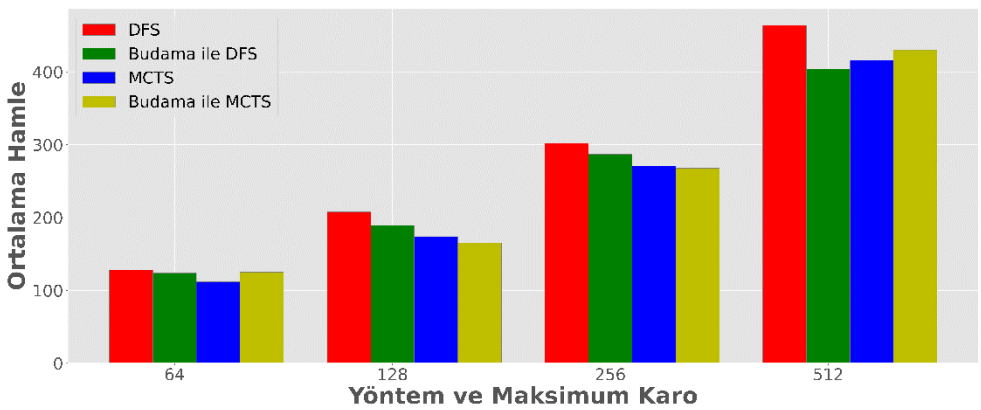
Şekil 10. Yöntemler Ortalama Oyun Süreleri

Oyundaki diğer bir başarı kriteri olan skor değeri dikkate alındığında Şekil 11'deki sonuçlar elde edilmiştir. Tablodaki sonuçlara bakıldığında aslında Şekil 9'deki sonuçlar ile benzerlik gösterdiği görülmektedir. Başka bir deyişle yöntemin elde ettiği maksimum karo değeri ne kadar yüksek ise o yöntem maksimum skor değerinde o derece yüksek başarı göstermektedir.



Şekil 11.Yöntemler ve Ortalama Skor

Şekil 12'de yöntemlerin elde ettikleri maksimum karo değerine göre ortalama hamle sayılarının grafiği verilmiştir. Bu grafiğe bakarak elde edilecek olan maksimum karo değeri için yapılacak olan hamle sayısının kullanılan yönteme bağlı olmadığı sonucu çıkarılabilir. Çünkü farklı yöntemlerin aynı maksimum karoya ulaşmak için yaptıkları hamle sayıları arasında dikkate değer bir fark bulunmamaktadır.



Şekil 12.Yöntemlerin Maksimum Karoya Göre Ortalama Hamle Sayısı

Giriş bölümünde verilen çalışmalar incelendiğinde bu çalışmaların ön işlem kısmında gerçekleştirdiği adımlar, kullanılan karmaşık ve ağır öğrenme süreçleri, bu süreçler için gerekli olan donanım ihtiyaçlarının karşılanamaması, aynı çalışma ortamının sunulamaması vb. durumları nedeniyle literatürde sunulan diğer yöntemler ile çalışmada verilen yöntemlerin aynı şartlar altında karşılaştırılması mümkün olmamaktadır. Ayrıca literatürde bulunan çalışmalarda da bu durum görülmüştür. Çalışmalar sadece kendi kullandıkları yöntemler için bir karşılaştırma sunabilmiş ve farklı çalışmalarda yer alan yöntemler ile karşılaştırma yapamamıştır.

Bu çalışmada ise ekstra bir donanım gerekmeden kişisel bir bilgisayar üzerinden 2048 oyunu temel öğrenme ve arama algoritmaları ile uygulanmış ve bu algoritmaların farklı metriklere göre performans karşılaştırmaları gerçekleştirilmiştir. Bu yöntemlere uygulanan budama işlemlerinin başarı performansını olumsuz yönde etkilediğini gözlenmektedir. Yapılan işlemin olumsuz bir sonucu olmasına rağmen kullanılmasının sebebi iş yükünü azaltmasıdır. Böylece Şekil 10 'da verilen grafikte görüldüğü üzere zamandan kazanç sağlanmıştır.

SONUÇ

Bu çalışmada 2048 oyunu için farklı algoritmalar kullanılarak optimal bir şekilde oynanması amaçlanmıştır. Bu amaç doğrultusunda DFS, MCTS ve Q-öğrenme algoritmaları kullanılmıştır. Kullanılan algoritmaların test edilmesi için her bir yöntem 100 defa çalıştırılmıştır. Yöntemlerin karşılaştırılmasında zaman, skor, maksimum karo değeri ve hamle sayısı şeklinde dört farklı metrik kullanılmıştır. Öğrenme tabanlı Q-öğrenme algoritması, yüksek sayıda oyun tekrarına rağmen olası durumların sadece küçük bir bölümünü gözlemleyebilmiş ve öğrenme sürecinde yetersiz kalmıştır. Arama tabanlı yöntemler skor ve maksimum karo metriklerinde iyi başarı gösterirken zaman metriği dikkate alındığında iyi sonuçlar verememiştir. Arama tabanlı yöntemlerde zaman metriği göz ardı edildiğinde en başarılı yöntem DFS olarak belirlenmiştir. Ağaç tabanlı algoritmalarda daha yüksek derinlik değerlerinin kullanılması daha etkili skor değerlerine ulaşabilmeyi mümkün kılmaktadır. Ağaç tabanlı yöntemlerde kullanılan budama işlemini daha farklı biçimlerde kullanmak iş yükünden ve zamandan kazanç sağlanabilmektedir.

Gelecek çalışmalar için, kullanılan arama algoritmaları farklı yöntemler ile desteklenerek performans iyileştirmeleri yapılabilir ve budama yönteminin zaman konusundaki avantajından yararlanmak için farklı budama yöntemleri test edilebilir. Ayrıca dinamik ve değişken sayıda derinlik değerleri kullanılarak zaman, bellek ve performans değerlendirmeleri gerçekleştirilebilir.

KAYNAKÇA

Boris, T., & Goran, S. (2017). Evolving neural network to play game 2048. *24th Telecommunications Forum, TELFOR 2016*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/TELFOR.2016.7818911

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., … Colton, S. (2012, March). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, pp. 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

Campbell, M., Hoane, A. J., & Hsu, F.-H. (2002). Deep Blue. In *Artificial Intelligence* (Vol. 134).

Cirulli G. (2014). The Creator of the 2048 Game . https://github.com/gabrielecirulli/2048 *improving reinforcement learning performance*. (n.d.).

Jacobsen, E. J., Greve, R., & Togelius, J. (2014). Monte mario: Platforming with MCTS. *GECCO 2014 - Proceedings of the 2014 Genetic and Evolutionary Computation Conference*, 293–300. Association for Computing Machinery. https://doi.org/10.1145/2576768.2598392

Jaśkowski, W. (2018). Mastering 2048 with delayed temporal coherence learning, multistage weight promotion, redundant encoding, and carousel shaping. *IEEE Transactions on Games*, *10*(1), 3–14. https://doi.org/10.1109/TCIAIG.2017.2651887

Kondo, N., & Matsuzaki, K. (2019). Playing game 2048 with deep convolutional neural networks trained by supervised learning. *Journal of Information Processing*, *27*, 340–347. https://doi.org/10.2197/ipsjjip.27.340

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. Retrieved from http://arxiv.org/abs/2005.01643

Lipowski, A., & Lipowska, D. (2011). *Roulette-wheel selection via stochastic acceptance*. https://doi.org/10.1016/j.physa.2011.12.004

Matsuzaki, K. (2017). Systematic selection of N-tuple networks with consideration of interinfluence for game 2048. *TAAI 2016 - 2016 Conference on Technologies and Applications of Artificial Intelligence, Proceedings*, 186–193. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/TAAI.2016.7880154

Mehta, R. (2014). *2048 is (PSPACE) Hard, but Sometimes Easy*. Retrieved from http://arxiv.org/abs/1408.6315

Newborn, M. (2000). Deep Blue's contribution to AI. In *Annals of Mathematics and Artificial Intelligence* (Vol. 28).

Nie, Y., Hou, W., & An, Y. (n.d.). *AI Plays 2048*.

Rodgers, P., & Levine, J. (n.d.). *An Investigation into 2048 AI Strategies*. Retrieved from http://www.veewo.com/games/

Sauren K, Jansen N, and Strüber D. (2022). Improving Reinforcement Learning Performance in 2048 Using Expert Knowledge.

Siau, K., & Wang, W. (2018). *Building Trust in Artificial Intelligence, Machine Learning, and Robotics Supply Chain Management View project*. Retrieved from www.cutter.com

Tong, S. (n.d.). *Roulette Wheel Selection Game Player*. Retrieved from https://digitalcommons.macalester.edu/mathcs_honors/30

Wang, F. Y., Zhang, J. J., Zheng, X., Wang, X., Yuan, Y., Dai, X., … Yang, L. (2016). Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, *3*(2), 113–120. https://doi.org/10.1109/JAS.2016.7471613

Weikai, W., & Kiminori, M. (n.d.). *Improving DNN-based 2048 Players with Global Embedding*. Retrieved from https://github.com/wwk1397/Improving-DNN-

Yeh, Kun Hao, I. Chen Wu, Chu Hsuan Hsueh, Chia Chuan Chang, Chao Chin Liang, and Han Chiang. (2017). "Multistage Temporal Difference Learning for 2048-like Games." IEEE Transactions on Computational Intelligence and AI in Games 9(4):369–80. doi: 10.1109/TCIAIG.2016.2593710.

Yu, Haofeng. (2016). From Deep Blue to DeepMind: What AlphaGo Tells Us.