# Best-First Search ve A* Search with Forward Checking Yöntemlerinin Karşılaştırılması: Yılan Oyununda Zayıf Yapay Zeka Oluşturulması

Ahmet Eren BOYACI

YAP 441 Yapay Us Dönem Projesi

*TOBB Ekonomi ve Teknoloji Üniversitesi*

aboyaci@etu.edu.tr

https://github.com/aeboyaci/BIL441-Project

https://youtu.be/wROF5xaXiMM

***Abstract* — Bu projede, Snake (Yılan) oyunu için Best-First Search ve A* Search with Forward Checking algoritmaları kullanılarak, otomatik bir aracı geliştirildi. Heuristik fonksiyon aracılığı ile değer hesaplanırken Manhattan uzaklığı kullanıldı. Proje, zayıf yapay zeka oluşturmayı amaçladı ve algoritmaların performansını karşılaştırmak için her iki yöntem de uygulandı. Projenin sonuçları, A* algoritmasının Best-First Search algoritmasına göre daha iyi bir performans sergilediğini gösterdi. Bu makale, Best-First Search ve A* algoritmalarının detaylı açıklamalarını içermektedir ve Yılan oyunu için bir aracının nasıl geliştirilebileceğini göstermektedir.**

***Keywords* — Zayıf Yapay Zeka, Bilgili Arama Algoritmaları, Best-First Search, A* Search, Öngörü (Forward-Checking)**

## I. GIRIŞ

Yapay zeka günümüzde birçok alanda hızla gelişmektedir. Yapay zeka teknolojileri, çeşitli zorlu görevleri gerçekleştirebilen, insan zekasına benzer kararlar verebilen ve insanların yaşam kalitesini artıran uygulamalar sunan birçok alanda kullanılmaktadır. Yapay zeka algoritmaları, özellikle oyunlar, robotik ve otomotiv sektörleri gibi alanlarda yaygın olarak kullanılmaktadır.

Bu proje, zayıf yapay zeka aracılığı ile bir yılan oyunun bilgisayar tarafından otomatik şekilde oynanması üzerine odaklanmaktadır. Yılan oyunu, klasik bir oyun olmasına rağmen, yapay zeka teknolojileri ile entegre edilerek, zayıf yapay zeka tarafından daha iyi ve daha hızlı bir şekilde oynanabilen bir oyun haline getirilmiştir. Projenin amacı, insan oyuncudan daha iyi ve daha hızlı bir şekilde oynayan bir zayıf yapay zeka geliştirmektir.

Proje, *Best-First Search*[1] ve *A* Search*[2] algoritmaları kullanılarak gerçekleştirilmiştir. Her iki algoritma da oyunun performans ölçütlerine göre seçilmiştir. Oyuncular, oyunu oynamak için bu iki algoritma arasından seçim yapabilirler.

Bu makalede, oyunun tasarımı ve uygulaması, kullanılan algoritmalar ve A* Search algoritmasındaki özel *Forward-Checking*[3] mekanizması ele alınmaktadır. Ayrıca, her iki algoritmanın performanslarının karşılaştırmalı bir analizi sunulmaktadır. Sonuçlar, özel forward-checking mekanizması ile A* Search algoritmasının, Best-First Search algoritmasına göre daha az düğümün keşfedilmesi ve optimal yolun bulunması için daha kısa süre gerektirdiğini göstermektedir.

Bu proje, arama algoritmalarının yapay zeka destekli oyunların geliştirilmesindeki etkinliğini göstermektedir ve belirli görevler için algoritmaların özelleştirilmesinin optimal performans elde etmek için önemini vurgulamaktadır. Ayrıca, bu proje, zayıf yapay zeka teknolojilerinin geliştirilmesi ve iyileştirilmesinde faydalı bir örnek oluşturmaktadır.

## II. YILAN OYUNU

### A. Oyun Açıklaması

Snake (Yılan), oyuncunun boyca uzayan bir çizgiyi yönlendirdiği konsepte sahip video oyunlarının genel adıdır.

Oyuncu, bir yılanı andıran, genellikle bir duvar boyunca (genellikle duvarlarla sınırlanmış), yiyecek toplayarak, kendi kuyruğu ve oyun alanının kenarları ile çarpışmayı önleyen bir yılana benzeyen uzun, ince bir yaratığı kontrol eder.

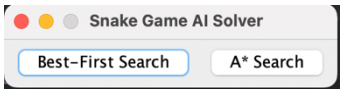
Her defasında bir yılan bir parça yiyecek yer, daha uzun olur ve bu da oyunu yavaş yavaş karmaşıklaştırır. Oyuncu yılanın kafasının hareket yönünü kontrol eder (genellikle 4 yön: yukarı, aşağı, sola, sağa) ve yılanın kuyruğu bir sonraki hareket eder. Oyuncu yılanın hareketini durduramaz.

### B. Oyun İmplementasyonu

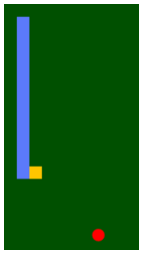
Aracının hareketlerinin net bir şekilde takip edilebilmesi adına yılan oyunu için *GUI (Graphical User Interface, Kullanıcı Arayüzü)* geliştirilmiştir. İlgili arayüz ve zayıf yapay zeka Java programlama dili kullanılarak geliştirilmiş ve aşağıdaki *Class (Sınıf)* yapılarından oluşmaktadır:

1. Main: Oyunun giriş (*main*) fonksiyonun bulunduğu ve oyunun çalıştırılmasını sağlayan sınıftır.
2. HomeGraphics: Zayıf yapay zekanın hangi bilgili arama yöntemi kullanarak hareket edeceğinin gözlem yapacak

olan kullanıcıya sorulduğu ekranı oluşturan sınıftır.




3. GameGraphics: Oyun tahtasının gösterildiği paneli oluşturan sınıftır.
4. Game: Oyun tahtasını çizen/gösteren ve yılanın hareketlerinin tahta üzerinde gösterilmesini sağlayan sınıftır.



5. GameRunner: Yılanın hareketlerinin *Game* sınfı içerisinde çizilmesi için *Thread* oluşturan sınıftır.
6. Node: Tahta üzerindeki konumların (x-y ikililerin) gösterilmesi ve işlenebilmesi için oluşturulan sınıftır.
7. FoodProvider: Yılan yemesi için tahta üzerinde elma oluşturulmasını sağlayan sınıftır.
8. Snake: Yılan ilgili bilgilerin tutulduğu ve yılanın hareket etmesi sağlayan fonksiyonlarun bulunduğu sınıftır.
9. AI: Best-First Search, A* Search with Forward Checking yöntemleri kullanılarak zayıf yapay zeka aracısının oluşturulmasını sağlayan sınıftır.

### C. *Oyun Akışı ve Çalışma Düzeneği*

Oyun; gözlem yapacak olan kullanıcının, aracının hangi yöntem (Best-First Search veya A* Search) ile zayıf yapay zekayı kullanacağını seçebileceği bir arayüzü ekranı ile başlamaktadır. Kullanıcı yöntem seçtikten sonra oyun tahtası arayüzü açılmakta ve bilgisayar otomatik şekilde yılanı hareket ettirerek oyunu zayıf yapay zeka aracılığı ile oynamaktadır. İlgili yöntemler ile normal bir insan kullancısından daha iyi bir sonuç elde edilse dahi kusursuz bir sonuç elde edilememekte ve bilgisayar yandığı (yılanın başı kendi bir parçasına değdiği) zaman oyun sonlanmaktadır. Oyun sonlandıktan sonra gözlem yapan kullanıcı ENTER tuşuna bastıktan sonra tekrardan başta çıkan arayüz çıkmaktadır.

## III. Çevre

Yapay zekanın performans ölçütünün ne olduğu ve nasıl bir ortamda bulunduğu, operasyonları anlayabilmek adına ve karşılaştırma yapabilmek için önemlidir. Bu doğrultuda, yılan oyunun bilgisayar tarafından otomatik şekilde oynanabilmesi için geliştirilen zayıf yapay zeka aracısı için yapılan PEAS analizi yapılmış ve çevre özellikleri incelenmiştir. İlgili analizler aşağıdaki gibidir:

### ***PEAS***[4] **analizi**

1. Performans Ölçütü: Yılan tarafından yenilen elma sayısı
2. Eyleyiciler: Kuzey/Doğu/Güney/Batı olmak üzere 4 farklı yöne hareket edilmesini sağlayan fonksiyon
3. Alıcılar: Oyun tahtasındaki elmaların yerini ve uzaklığını hesaplayan fonksiyonlar
4. Çevre: Oyun tahtası

### Çevre özellikleri

1. Gözlem: Tam gözlenebilir
   Aracı oluşan elmaların nerede olduğunu net bir şekilde tespit edebilmektedir.
2. Durum: Rastgele
   Elmaların yerleri rastgeledir, bir sonraki durum tahmin edilemez
3. Süreç: Takip eden
   Yılanın boyutu elma yedikçe uzamaktadır ve yılanın herhangi bir parçası diğer bir parçasına değdiği zaman oyun bitmektedir. Dolayısıyla, daha önceki durumlar anlık durumu etkilemektedir.
4. Değer: Sürekli
5. Aracı: Tek aracı
6. Değişim: Statik

## IV. Bilgili Arama

Heuristik fonksiyonları, yapay zeka algoritmaları için önemli bir araçtır ve arama alanının boyutunu azaltarak, arama süresini önemli ölçüde azaltabilirler. İyi bir heuristik fonksiyonu, arama maliyetini azaltırken, arama başarısını da arttırır.

Heuristik fonksiyonun seçimi algoritmanın başarılı ve iyi sonuç verebilmesi adına kritik bir önem taşır. Gerçek maliyet değerine yakın ancak daha küçük değerler veren bir fonksiyon seçmek kullanılacak heuristik algoritmalarının sonuçlarını iyileştirmektedir. Bu doğrultuda, proje kapsamında kullanılan algoritmalarda *Manhattan Distance*[5] değeri hesaplayan bir fonksiyon seçilmiştir.

Best-First Search algoritması, bir heuristik fonksiyon kullanarak, arama maliyetini azaltır ve en iyi hamleyi seçer. Bu sayede, en kısa yolu bulmak için etkili bir yöntemdir. A* algoritması ise, Best-First Search algoritmasına benzer şekilde çalışır, ancak hedefe ulaşmak için öngörülen toplam maliyeti minimize eder. Bu nedenle, A* algoritması, Best-First Search algoritmasından daha iyi sonuçlar verebilir.

İyi bir heuristik fonksiyonu seçmek, algoritmanın performansını artırabilir. Bu fonksiyon, arama alanının boyutunu azaltmak ve maliyeti düşürmek için kullanılabilir. Bu proje için, Manhattan uzaklığı bir heuristik fonksiyon olarak seçildi. Bu fonksiyon, hedefe olan en kısa mesafeyi hesaplamak için kullanıldı ve Best-First Search ve A* algoritmalarının performansını artırdı.

İyi bir heuristik fonksiyonu seçmek, algoritmanın performansını artırabilir ve arama süresini önemli ölçüde azaltabilir. Best-First Search ve A* algoritmaları, bu projede seçilen en uygun algoritmalar oldu ve heuristik fonksiyonlarının kullanımı ile beraber, yılan oyununu otomatik olarak oynatabilecek bir yapay zeka geliştirildi.

## V. Zayıf Yapay Zeka Algoritması ve Forward-Checking Mekanizması

Best-First Search seçeneği seçildiyse, heuristik değer olarak Manhattan uzaklığı hesaplanmakta ve buna göre bir sonraki hareket (Kuzey/Doğu/Güney/Batı) seçilmektedir. Eğer bilgili arama yöntemi ile bir karar verilememiş ise rastgele bir hareket seçilmektedir.

A* Search seçildiyse, yine heuristik değer olarak Manhattan uzaklığı hesaplanmaktadır. Ancak bu seçenekte sanal bir yılan oluşturulmakta ve gerçek yılan hareket etmeden önce bazı öngörüler oluşturulmaktadır. Öngörü hesaplamaları yapılırken sanal olarak oluşturulan veya gerçek yılan kendi kuyruğunu yiyebilecek mi kontrol edilmektedir. Şayet bu durum gerçekleşebiliyorsa; yılanın başı, kuyruğun hareket etmeden önce bulunduğu noktaya gitmeye başladığında kuyruk ve diğer parçalar da hareket edeceği için yanmak imkansız hale gelmektedir. Bu kapsamda öngörüler aşağıdaki gibi oluşturulmuştur:

1. Eğer sanal yılan, yanmadan elmayı yiyemeyecek ise gerçek yılan anlık konumundaki kuyruğunu yiyebilecek mi kontrol edilmektedir. Gerçek yılan kuyruğunu yiyebilecekse A* Search ile oluşturulan rotayı izlemekte, yiyemeyecekse rastgele hesaplanan hareketi yapmaktadır.
2. Eğer sanal yılan, yanmadan elmayı yiyebilecek ise sanal yılan hareket ettirilerek elmayı yiyebilecek noktaya götürülmekedir. Bu noktaya geldikten sonra, sanal yılan kendi kuyruğunu yiyebilecek mi kontrol edilmektedir. Sanal yılan kuyruğunu yiyebilecekse A* Search ile oluşturulan rotayı izlemekte, yiyemeyecekse gerçek yılan kendi kuyruğunu yiyebilecek mi diye kontrol edildikten sonra eğer yiyebiliyorsa kuyruğunu yiyecek şekilde, yiyemiyorsa direkt elmaya gidecek şekilde hareket etmektedir.

## VI. Karşılaştırma

Genel bir karşılaştırma yapılabilmesi adına uygulama birden çok kez çalıştırılmış ve aracının; Best-First Search yöntemi kullanması durumunda ortalama olarak ~35-60, A* with Forward Checking yöntemi kullanması durumunda ise ortalama olarak ~85-120 elma yiyebildiği gözlemlenmiştir.

Raporda örnek bir sonuç sunulması için çalıştırıldığında:

**Best-First Search**


**A* Search with Forward Checking (arayüz menüsündeki A* Search)**


sonuçları elde edilmiştir.

Best-First Search seçeneğinde, *Heap Overflow*[6] (*java.lang.OutOfMemoryError: Java heap space*) hatası ile karşılaşıldığı için arama ağacı için bir derinlik kısıtı (anlık durumda 8) eklenmiştir. Bu derinlik kısıtı arttırıldığında ise yenilen elma sayısı artarken, hafıza hataları ortaya çıkarttığı ve yılanın hareketlerinin yavaşladığı gözlemlenmiştir. Söz konusu durum A* Search tabanlı yöntemde oluşmamaktadır.

Sonuç olarak, A* Search tabanlı yöntemde hem bilgili arama algoritmasının hem de oluşturulan öngörülerin performansa ciddi etkisi olduğu tespit edilmiştir.

## Açıklamalar

[1] Best-First Search, genişletilmiş arama algoritmalarından biridir ve bir graf veya ağaç yapısı üzerinde arama yapmak için kullanılır. Çözüm alanındaki her düğüm için bir tahmin yaparak en iyi hamleyi seçer ve bu hamle doğrultusunda ilerler. Bu sayede, her zaman en iyi hamleyi yapmak için hareket eder.

Best-First Search algoritmasında, genişletilmesi gereken düğümler bir öncelik sırasına göre sıralanır. Bu sıralama, bir heuristik fonksiyon kullanılarak yapılır. Heuristik fonksiyon, Best-First Search'ün ilerlemesi sırasında kullanılan bir fonksiyondur ve her durum için bir tahmin yapar. Manhattan uzaklığı, bir heuristik fonksiyon olarak kullanılabilecek bir örnektir. Bu fonksiyon, bir düğümün hedefe olan en kısa mesafeyi hesaplar ve düğümler bu mesafeye göre öncelik sırasına sokulur.

Best-First Search algoritması, hedefe ulaşana kadar genişletilmemiş en öncelikli düğümü genişletir ve böylece en iyi hamleler dizisini bulur. Bu nedenle, Best-First Search, genellikle en kısa yolu bulmak için kullanılır ve yolun optimizasyonu için etkili bir araçtır.

Manhattan uzaklığı, iki nokta arasındaki en kısa yolu hesaplamak için kullanılan bir metrik fonksiyondur. Bu fonksiyon, iki noktanın x ve y koordinatları arasındaki farkları toplar ve toplamı Manhattan uzaklığı olarak adlandırır. Best-First Search algoritmasında, Manhattan uzaklığı, hedefe olan en kısa mesafeyi hesaplamak için kullanılan bir heuristik fonksiyon olarak kullanılabilir.

[2] A* algoritması, bir heuristik fonksiyon kullanarak, hedefe en kısa yolu bulan bir arama algoritmasıdır. Algoritmanın çalışma prensibi, Best-First Search algoritmasına benzerdir, ancak öngörülen toplam maliyeti minimize ederek hedefe ulaşır. Bu öngörülen toplam maliyet, o anki maliyetin ve hedefe olan uzaklığın toplamıdır.

Algoritma, iki farklı liste kullanır: açık liste (open list) ve kapalı liste (closed list). Açık liste, arama sırasında ziyaret edilen ve henüz hedefe ulaşmayan düğümleri içerirken, kapalı liste, zaten ziyaret edilen ve hedefe ulaşan düğümleri içerir.

A* algoritması, başlangıç düğümünden hedef düğüme doğru ilerlerken, açık listeye yeni düğümler ekler ve mevcut düğümleri kapatır. Daha sonra, açık listedeki düğümleri toplam maliyetlerine göre sıralar ve en düşük maliyete sahip düğümü seçer. Seçilen düğüm, hedef düğüm olana kadar bu işlemi tekrarlar.

A* algoritması, bir heuristik fonksiyon kullanarak, arama alanının boyutunu azaltır ve en kısa yolu bulur. Ancak, doğru bir heuristik fonksiyon seçmek, performansı önemli ölçüde etkileyebilir. Proje için, Manhattan uzaklığı bir heuristik fonksiyon olarak seçildi ve performansı artırdı.

[3] Forward-Checking, bir arama ağacında belirli bir yolun ilerlemesinde bir değişkenin değeri atanmadan önce olası değerleri düşünerek çakışmaları önlemek için kullanılan bir tekniktir. Bu teknik, Best-First Search ve A* Search gibi arama algoritmalarında da kullanılabilir. Daha doğru bir maliyet tahmini yapılmasını sağlar ve arama sırasında daha az geri dönüş yapılmasına olanak tanır.

[4] PEAS analizi yapay zeka sistemlerinin tasarımında kullanılan bir yöntemdir ve sistemlerin problemi çözmek için nasıl hareket etmeleri

gerektiği konusunda bir çerçeve sağlar. PEAS kısaltması, Problem, Environment, Actuators (Eyleyiciler), ve Sensors (Sensörler) kelimelerinin baş harflerinden oluşur.

- Problem (P): Sistem tarafından çözülmeye çalışılan görevi ifade eder.
- Environment (E): Sistem ve görevi arasındaki bağlantıyı ifade eder ve genellikle bir durum uzayı veya fiziksel bir ortamdır.
- Actuators (A): Sistem tarafından kullanılan eyleyicilerdir. Bu, sistem tarafından yapılan eylemleri ifade eder, örneğin bir robotun hareketi gibi.
- Sensors (S): Sistem tarafından kullanılan sensörlerdir. Bu, sistemin çevreye nasıl tepki verdiğini ve çevreyi nasıl algıladığını ifade eder, örneğin bir robotun kamerası veya dokunmatik sensörü gibi.

[5] Manhattan uzaklığı, iki nokta arasındaki doğrusal mesafeyi ifade eden bir ölçüttür. İki nokta arasındaki Manhattan uzaklığı, noktaların yatay ve dikey koordinatları arasındaki farkların toplamıdır. Örneğin, (x1, y1) ve (x2, y2) noktaları arasındaki Manhattan uzaklığı, |x2 - x1| + |y2 - y1| şeklinde hesaplanır. Manhattan uzaklığı, özellikle oyunlarda ve yol tarifi gibi uygulamalarda kullanılır. Ayrıca A* arama gibi algoritmaların heuristik fonksiyonları olarak da sıklıkla tercih edilir.

[6] Genellikle bu hata, Java yığınında bir nesneyi ayırmak için yeterli alan olmadığında atılır. Bu durumda, çöp toplayıcı yeni bir nesneyi barındırmak için yer açamaz ve yığın daha fazla genişletilemez. Ayrıca, bir Java sınıfının yüklenmesini desteklemek için yerel bellek yetersiz olduğunda bu hata atılabilir. Nadir bir durumda, çöp toplama işlemi için aşırı miktarda zaman harcandığında ve çok az bellek boşaltıldığında bir java.lang.OutOfMemoryError atılabilir.

3.2 Understand the OutOfMemoryError Exception, Oracle Official Documentation, https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/memleaks002.html