**8 Taş Oyun Tasarımı ve A\* Algoritması ile Oyunun Çözüm Uygulaması**

**Hazırlayan**

Fırat Kaan Bitmez

**Öğrenci Numarası**

23281855

**Dersin Hocası**

Doç.Dr. Gökhan Kayhan

**Giriş**  
**A\* algoritması**, yol bulma problemlerini çözmek için kullanılan etkili bir arama algoritmasıdır. Ancak, **8 taş oyunu** gibi durum tabanlı oyunlar için kullanmak uygun olmayabilir çünkü bu tür oyunlar genellikle çok daha karmaşık bir durum uzayına sahiptir. Fakat biz Yapay Zeka dersi kapsamında 8 taş oyununu tasarlayıp A\* algoritması çözümleyip anlık olarak durumunu izleyebileceğimiz bir tasarım ve kodlama yapacağız. Normalde bu tarz durum uzayına sahip oyunlarda genellikle minmax algoritmasını kullanmak sizin için daha faydalı olacaktır. Bu Kodlamayı yaparken hem basit ve kolay olması hemde web tabanlı olarak yayınlanabilmesi için **HTML**,**CSS** ve **Javascript** kullanacağız bu sayede kodlamamız basit olacak hemde daha büyük bir kitle anlayabilecek.

**Kullanılan Kütüphaneler ve Araçlar**

**Boostrap** : Görsel ve Dinamik olarak daha iyi bir görsel tasarım vermek için boostrap kütüphanesi kulanıldı.

**HTML,CSS,Javascript** : Kodlama

**A\* Algoritması**

Tasarıma Başlamadan Önce A\* (A-star) Algoritmasını kısaca anlatmam gerekirse , Sezgisel bilgilerin kullanımında değer fonksiyonun belirlenmesi büyük önem taşımaktadır. Bu fonksiyon durumları ifade eden çözüm grafı üzerinde verilmiş başlangıç durumundan hedefe ulaşmak için gereken minimum yolun bulunmasında daha az tarama yapmaya olanak tanımalıdır.

Eğer başlangıç **S** düğümünden **x** düğümüne kadar olan yolu **g(x)** ile, **x** düğümünden ise **f** düğümüne olan yol değerine **h(x)** ile ifade edersek A\* algoritması denilen sezgisel arama için aşağıdaki bağlantıyı elde ederiz.

**F(x)=g(x)+h(x)**

Burada **g(x)** , x durumunun gerçek olan o anki değeridir. **H(x)** ise x düğümünden çözüme olan gidişlerin sezgisel değeridir. Yukarıda verilen **g(x)** ve **h(x)** fonksiyonlarının tanımları çözüme en az aramayla yaklaşım açısından çok önemlidir.

**8 Taş Problemi A\* Algoritması**

8 Taş probleminde başlangıç durumundan hedef durumuna ulaşmak için gereken minimum **n** hareket sayısına ilişkin problemin algoritma karmaşıklığı **O(3n)**  şeklinde belirlenir.

Problemin çözümünü ağaç şeklinde ifade edersek **g(x) değeri başlangıçtan x düğümüne kadar olan derinliği ifade edecektir**. **H(x) değerlendirmesi olarak ise kendi yerinde olmayan taşların sayısını ele alalım.**

Bileşenleri bu şekilde belirledikten sonra minimum f(x) değerine göre açılması gereken düğüm seçilmektedir.

G(x) > Derinlik

H(x) > Hedef durumda yerinde olmayan taşların sayısı

**metin, diyagram, taslak, plan içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Kodlamalar**

Bu bölümde Sadece kodlamanın önemli bölümüne değinilecektir. Tüm kodlamayı görmek için <https://github.com/firatkaanbitmez/yapay-zeka> Site adresine giderek **Proje kaynak kodu**na ulaşabilirsiniz.

  // Oyun tahtasını oluştur

    for (var i = 0; i < 9; i++) {

        var tile = $('<div class="tile"></div>').attr('data-index', i); // Her bir kutucuk için div oluştur

        tile.click(moveTile); // Kutucuğa tıklanınca hareket fonksiyonunu çağır

        tiles.push(tile); // Kutucukları diziye ekle

        board.append(tile); // Kutucukları tahtaya ekle

    }

    // Başlangıç durumunu belirle

    var numbers = Array.from({ length: 9 }, (\_, i) => i === 8 ? '' : (i + 1)); // 1'den 9'a kadar sayıları oluştur, sonuncuyu boş bırak

    renderBoard(); // Tahtayı ilk kez çiz

 // Kareye tıklandığında hareketi işle

    function moveTile() {

        var clickedIndex = parseInt($(this).attr('data-index')); // Tıklanan kutucuğun indeksini al

        if (isValidMove(clickedIndex)) { // Geçerli bir hareket mi kontrol et

            var temp = numbers[emptyIndex]; // Geçici değişken kullanarak sayıları yer değiştir

            numbers[emptyIndex] = numbers[clickedIndex];

            numbers[clickedIndex] = temp;

            emptyIndex = clickedIndex; // Boş kutucuğun indeksini güncelle

            renderBoard(); // Tahtayı yeniden çiz

            if (isGameWon()) {

                alert("Tebrikler, oyunu kazandınız!"); // Oyun kazanıldıysa tebrik et

            }

        }

    }

// Hedef duruma ulaşmak için gereken sezgisel maliyeti hesapla

    function calculateHeuristic(state) {

        var misplaced = 0;

        for (var i = 0; i < state.length; i++) {

            if (state[i] !== '' && state[i] !== i + 1) {

                misplaced++;

            }

        }

        return misplaced; // Yanlış yerde olan kutucuk sayısını döndür

    }

    // A\* algoritması ile 8 taş problemi çözümü

    function solveeightstone(eightstone) {

        var goalState = [1, 2, 3, 4, 5, 6, 7, 8, '']; // Hedef durumu belirle

        var startNode = { // Başlangıç düğümünü oluştur

            state: eightstone,

            parent: null,

            move: null,

            cost: 0,

            heuristic: calculateHeuristic(eightstone)

        };

        openList = [startNode]; // Açık liste ile başla

        closedList = []; // Kapalı listeyi sıfırla

        while (openList.length > 0) { // Açık liste boş olana kadar devam et

            var currentNode = openList[0]; // En iyi düğümü seç (en düşük f + h maliyetli)

            var currentIndex = 0;

            openList.forEach(function (node, index) {

                if (node.cost + node.heuristic < currentNode.cost + currentNode.heuristic) {

                    currentNode = node;

                    currentIndex = index;

                }

            });

            if (currentNode.state.toString() === goalState.toString()) { // Çözümü bulduk mu?

                var solution = [];

                var current = currentNode;

                var realCost = 0;

                while (current !== null) {

                    if (current.move) {

                        solution.unshift(current.move);

                        realCost++;

                    }

                    current = current.parent;

                }

                var heuristicCost = currentNode.heuristic;

                return { solution: solution, realCost: realCost, heuristicCost: heuristicCost }; // Çözümü döndür

            }

            openList.splice(currentIndex, 1); // Açık listeden çıkar

            closedList.push(currentNode); // Kapalı listeye ekle

            var neighbors = generateNeighbors(currentNode); // Komşu düğümleri oluştur

            neighbors.forEach(function (neighbor) {

                if (!containsNode(closedList, neighbor.state)) {

                    var gScore = currentNode.cost + 1;

                    var inOpenList = containsNode(openList, neighbor.state);

                    if (!inOpenList || gScore < neighbor.cost) {

                        neighbor.cost = gScore;

                        neighbor.heuristic = calculateHeuristic(neighbor.state);

                        neighbor.parent = currentNode;

                        if (!inOpenList) {

                            openList.push(neighbor);

                        }

                    }

                }

            });

        }

        return null; // Çözüm bulunamadı

    }

// Komşu düğümleri oluştur

    function generateNeighbors(node) {

        var neighbors = [];

        var emptyIndex = node.state.indexOf('');

        var emptyRow = Math.floor(emptyIndex / 3);

        var emptyCol = emptyIndex % 3;

        var moves = [

            { row: -1, col: 0, action: 'Boş Kutucuk Yukarı' },

            { row: 1, col: 0, action: 'Boş Kutucuk Aşağı' },

            { row: 0, col: -1, action: 'Boş Kutucuk Sola' },

            { row: 0, col: 1, action: 'Boş Kutucuk Sağa' }

        ];

        moves.forEach(function (move) {

            var newRow = emptyRow + move.row;

            var newCol = emptyCol + move.col;

            if (newRow >= 0 && newRow < 3 && newCol >= 0 && newCol < 3) {

                var newState = node.state.slice();

                var newIndex = newRow \* 3 + newCol;

                newState[emptyIndex] = newState[newIndex];

                newState[newIndex] = '';

                var newHeuristic = calculateHeuristic(newState);

                neighbors.push({

                    state: newState,

                    parent: node,

                    move: move.action,

                    cost: 0,

                    heuristic: newHeuristic

                });

            }

        });

        return neighbors;

    }

// Çözüm bilgilerini göster

    function displaySolutionInfo(solution, realCost, heuristicCost, totalCost, solutionTree) {

        var solutionText = "Çözüm Adımları:<br>";

        for (var i = 0; i < solution.length; i++) {

            solutionText += solution[i] + "<br>"; // Çözüm adımlarını metin olarak oluştur

        }

        if (solutionTree) {

            solutionText += "<div class='solution-tree'><h2>Çözüm Ağacı:</h2><div class='node-details'><table><tr><th>Adım</th><th>Durum</th><th>Gerçek Maliyet</th><th>Sezgisel Maliyet</th><th>Toplam Maliyet</th></tr>";

            solutionTree.forEach(function (node, index) {

                solutionText += "<tr><td>" + index + "</td><td>" + renderBoardState(node.state) + "</td><td>" + node.cost + "</td><td>" + node.heuristic + "</td><td>" + (node.cost + node.heuristic) + "</td></tr>"; // Çözüm ağacını metin olarak oluştur

            });

            solutionText += "</table></div></div>";

        }

        $('#solutionInfo').html(solutionText); // Çözüm bilgilerini ekrana yazdır

    }

**Sonuç**

Sonuç olarak, bu projede 8 Taş Oyunu tasarlarmış ve çözümü için A\* algoritması kullanılarak bir çözüm uygulaması geliştirilmiştir. Projenin amacı, yapay zeka dersi kapsamında hem basit hem de interaktif bir uygulama tasarlamaktır. A\* algoritmasının temel prensipleri projede kullanılarak, kullanıcıya oyun tahtasını karıştırma, taşları hareket ettirme ve çözümü görsel olarak takip etme imkanı sunulmuştur. Bu proje, yapay zeka kavramlarını uygulamalı olarak anlamak ve öğrenmek için etkili bir araç olmuştur. Gelecekte, projenin daha karmaşık oyunlar için genişletilmesi veya görsel tasarımın iyileştirilmesi gibi geliştirme alanları bulunmaktadır.