T.C
DOKUZ EYLÜL ÜNİVERSİTESİ
FEN FAKÜLTESİ
BİLGİSAYAR BİLİMLERİ BÖLÜMÜ

UNİTY OYUN MOTORU İLE MAKİNE ÖĞRENMESİ

Olcay AKDAĞ

Cem ERDOĞAN

Danışman: DR. ÖĞR. Üyesi Ayşe Övgü KINAY

Mayıs, 2023

İZMİR

Olcay AKDAĞ ve Cem ERDOĞAN tarafından **DR. ÖĞR. Üyesi Ayşe Övgü KINAY** yönetiminde hazırlanan **"Unity Oyun Motoru ile Makine Öğrenmesi"** başlıklı rapor tarafımızca okunmuş, kapsamı ve niteliği açısından bir Bitirme Projesi olarak kabul edilmiştir.

DR. ÖĞR. Üyesi
Ayşe Övgü KINAY

ÖZET

Bu bitirme projemizin kapsamında, bir *match-3* oyununda kolay, orta ve zor seviyelerde tasarlanmış bölümleri pekiştirmeli öğrenme yöntemiyle test etmek ve doğruluğunu değerlendirmek için bir makine öğrenmesi modeli geliştirdik. Yaptığımız istatistiksel analizler sonucunda, makine öğrenmesi modelleri arasında anlamlı bir fark bulduk. Projemizde, test sürecini başarılı bir şekilde gerçekleştiren makine öğrenmesi modeli kullanarak kolay, orta ve zor seviyelerdeki bölümleri test ettik. Sonuçlarımız, bu seviyelerde farklı düzeylerdeki modellerin arasında istatistiksel olarak anlamlı bir fark olduğunu ortaya koydu. Bu da geliştirdiğimiz pekiştirmeli öğrenme yönteminin, farklı zorluk seviyelerindeki bölümleri başarıyla ayırt edebildiğini ve test edebildiğini gösterebilir. Bu bitirme projesi, *match-3* oyunlarında oyun tasarımının ve zorluk seviyelerinin pekiştirmeli öğrenme ile optimize edilebileceğini göstermek için yapılmıştır. Elde ettiğimiz sonuçlarla, oyun geliştiricilerinin, oyuncuların daha keyifli ve tatmin edici bir oyun deneyimi yaşamasını sağlamaları için bölüm tasarımlarını buna göre geliştirmelerine yardımcı olabilmek amaçlanmıştır.

Anahtar kelimeler: Mobil oyun, makine öğrenimi, pekiştirmeli öğrenme, unity oyun motoru.

ABSTRACT

In this study, we developed a machine learning model to test and evaluate levels designed as "easy", "medium", and "hard" in a *match-3* game using reinforcement learning. The t-test analysis we conducted revealed a statistically significant difference among the models. In our project, we successfully tested the levels using the machine learning model, which effectively performed the testing process. Our results demonstrated a statistically significant difference among the models at these levels, indicating that our developed reinforcement learning approach was capable of successfully distinguishing and testing levels with different difficulty levels. This thesis aimed to showcase that game design and difficulty levels in *match-3* games can be optimized using reinforcement learning. The obtained results were intended to assist game developers in enhancing their level designs to provide players with a more enjoyable and satisfying gaming experience.

Keywords: Mobile game, machine learning, reinforcement learning, unity game engine.

İÇİNDEKİLER

ÖZET 3
ABSTRACT 4
İÇİNDEKİLER 5
ŞEKİLLER DİZİNİ 6
1. GİRİŞ 7
2. PEKİŞTİRMELİ ÖĞRENME 8
2.1. Pekiştirmeli Öğrenmenin Avantajları 9
2.2. Pekiştirmeli Öğrenmenin Dezavantajları 9
2.3. Değerlendirici Geri bildirim 11
2.4. Eylem-Değer Yöntemleri 11
2.5. Ajan-Çevre Arayüzü 12
2.6. Ödüller ve Hedefler 13
3. MATCH-3 OYUNLAR 14
3.1. Motivasyon Unsurları 14
3.2. Bölüm Tasarımı ve Makine Öğrenimi 15
4. UYGULAMA 16
4.1. Geliştirme Ortamları 16
4.1.1. Unity 2021.3.11f1 16
4.1.2. C# Yazılım Dili 16
4.1.3. Visual Studio 16
4.1.4. Unity ML-Agents 2.3.0 16
4.1.5. Google Sheets 17
4.2. T-testi 17
4.3. Match-3 Oyunun Geliştirilmesi 18
4.4. Pekiştirmeli Öğrenmenin geliştirilmesi 21
4.5. Modellerin Test Edilmesi 22
5. SONUÇ VE DEĞERLENDİRME 27
KAYNAKÇA 28
EKLER 30

ŞEKİLLER DİZİNİ

Şekil 2.1 13
Şekil 4.1 20
Şekil 4.2 21
Şekil 4.3 22
Şekil 4.4 23
Şekil 4.5 24
Şekil 4.6 25
Şekil 4.7 25
Şekil 4.8 25
Şekil 4.9 26
Şekil 4.10 26
Şekil 4.11 26

1. GİRİŞ

Bu çalışma, *match-3* oyunlarındaki bölüm tasarımlarının istenilen zorluk seviyelerini karşılayıp karşılamadığını değerlendirmek için pekiştirmeli öğrenme ile yapay zekâ modellerinin kullanılmasını araştırmaktadır. *Match-3*, oyuncuların benzer renkte veya desende üç veya daha fazla nesneyi eşleştirmeye çalıştığı popüler bir oyun türüdür. Bu oyunlarda, bölüm tasarımlarının doğru zorluk seviyelerine sahip olması, oyuncuların oyunu keyifle oynamalarını ve motivasyonlarını yüksek tutmalarını sağlamaktadır. Pekiştirmeli öğrenme, yapay zekâ modellerinin bir ortamda deney yaparak ödüllendirme ve cezalandırma yoluyla en iyi stratejileri öğrenmesini sağlayan bir öğrenme yaklaşımıdır. Bu çalışmada, *match-3* oyunları için üç seviyeli yapay zekâ modelleri geliştirilmiştir: acemi, orta ve iyi seviyeler. Bu modeller, bölüm tasarımlarını test etmek ve istenilen zorluk seviyelerini değerlendirmek için kullanılmaktadır. Yapay zekâ modelleri, *match-3* oyunlarında tasarlanan bölümleri test ederek oyunun başarı oranını ve skorunu optimize etmek için pekiştirmeli öğrenme algoritmalarıyla eğitilmiştir. Acemi seviye modeli, daha kolay bölüm tasarımlarını çözmek için eğitilmiştir. Orta seviye modeli, orta zorluktaki bölüm tasarımlarını çözebilmektedir. İyi seviye modeli ise daha karmaşık ve zor bölüm tasarımlarını başarılı bir şekilde çözebilmektedir. Bu yapay zekâ modelleri, geliştiricilerin tasarladıkları bölümlerin istenilen zorluk seviyelerine uygun olup olmadığını test etmelerine olanak sağlamaktadır. Geliştiriciler, tasarladıkları bölümü bu yapay zekâ modelleri üzerinde çalıştırarak, modelin başarı oranını ve skorunu gözlemleyebilirler. Böylece, bölümün zorluk seviyesini istedikleri gibi ayarlayabilirler ve oyuncuların oyunu keyifle oynamalarını sağlayabilirler. Bu çalışma, *match-3* oyunları için pekiştirmeli öğrenme ile yapay zekâ modellerinin bölüm tasarımının değerlendirilmesinde nasıl kullanılabileceğini incelemektedir.

2. PEKİŞTİRMELİ ÖĞRENME

Pekiştirmeli öğrenmenin tarihi iki ana konuya sahiptir. Bunlardan ilki deneme yanılma ve hayvan psikolojisinden başlamıştır. Diğer konu, optimal kontrol problemini değer fonksiyonları ve dinamik programlama ile çözmeye dayanır ve öğrenmeyi içermemektedir. 1980'lerin sonlarına doğru bu iki konu birleşerek pekiştirmeli öğrenmenin gelişmesini sağlamıştır.

Pekiştirmeli öğrenme zeki ajan (agent) olarak isimlendirilen makinemiz, ortamda karşılaştığı durumlara tepkiler vererek karşılığında bir ödül veya ceza puanı alarak kümülatif ödülü maksimize etmeyi amaçlayan bir makine öğrenmesi yöntemidir. (WEB_1, 2019)

Pekiştirmeli öğrenme Markov karar süreci modelini kullanmaktadır. Markov karar süreci üç bölümden oluşur. Bunlar;

- Bir dizi olası dünya durumu S
- Bir dizi olası aksiyonlar A
- Gerçek değerli bir ödül fonksiyonu R (S, A)

(WEB_2, 2001)

Pekiştirmeli öğrenme süreci keşif ve sömürü olarak iki aşamadan oluşur. Ajan, keşif aşamasında gelecekte daha iyi karar vermesine yol açabilecek senaryolar hakkında bilgi toplar. Sömürü aşamasında ajan, bu edinilmiş bilgilerle mümkün olan en yüksek ödül puanını aradığına ulaşmaya çalışır.

(WEB_3, 2020)

Pekiştirmeli öğrenme üç temel ve bir opsiyonel öğeden oluşur. Bunlar;

- Politika; ajanın verilen zamandaki davranış biçimini tanımlar.
- Ödül fonksiyonu; ajanın aldığı aksiyon sonrasında aldığı ödül puanıdır.
- Değer fonksiyonu; ajanın aksiyonları sonucunda uzun vadede aldığı ödül puanını tanımlar. Burada önemli olan kısa vadede çok puanın uzun vadede çok puan anlamına gelmediği ve ilerleyen aşamalarda daha çok ödül puanı elde edebilmek için bazen kısa vadede düşük puanın seçilmesi gerekebildiğidir.
- Ve ek olarak model; çevre modeli ajanın bir sonraki durum ve ödül puanını tahmin etmesi için gerçek çevrenin bir simülasyonu gibidir.

2.1.Pekiştirmeli Öğrenmenin Avantajları

Pekiştirmeli öğrenmenin birçok avantajı vardır. İşte pekiştirmeli öğrenmenin bazı önemli avantajları;

1. Model Bağımsızlık: Pekiştirmeli öğrenme, bir modelin nasıl çalıştığını veya çevrenin ne şekilde değiştiğini tam olarak bilinmemesi durumunda bile kullanılabilir. Ajan, deneyimlerini çevreyle etkileşim halindeyken öğrenir ve bu deneyimlere dayanarak hareketlerini optimize etmeye çalışır. Bu, gerçek dünya problemlerinde kullanım kolaylığı sağlar.
2. Hedef Esnekliği: Pekiştirmeli öğrenmede ajanın hedefi, toplamda alacağı ödül puanını maksimize etmek veya bir belirli hedefe ulaşmaktır. Bu, farklı problemler ve hedefler için uyarlanabilme esnekliği sağlar. Ajan, farklı hedefleri optimize etmek için gerekli stratejileri öğrenebilir.
3. Deneyime Dayalı Öğrenme: Pekiştirmeli öğrenme, deneyimlerden öğrenmeye dayanır. Ajan, çevreyle etkileşim halindeyken geri bildirim alır ve bu geri bildirimleri kullanarak stratejisini günceller. Bu, ajanın gerçek dünyadaki dinamikler ve belirsizliklerle başa çıkabilmesini sağlar.
4. Sürekli Öğrenme: Pekiştirmeli öğrenme, sürekli olarak yeni bilgilerle beslenebilir ve mevcut stratejisini güncelleyebilir. Ajan, çevreyle etkileşim halinde olduğu sürece öğrenmeye devam edebilir ve değişen çevre koşullarına uyum sağlayabilir.
5. Keşif ve Sömürü Dengesi: Pekiştirmeli öğrenme, keşif ve sömürü arasında denge kurmayı sağlar. Ajan, keşif yaparak yeni bilgiler edinir ve keşif sırasında risk alırken, aynı zamanda mevcut bilgilere dayanarak istikrarlı bir strateji geliştirmeye çalışır. Bu denge, ajanın hem yeni fırsatları keşfetmesini sağlar hem de daha önceden öğrenilmiş bilgileri kullanarak başarı sağlar.
6. Genelleme Yeteneği: Pekiştirmeli öğrenme, genelleme yeteneği sağlayabilir. Ajan, deneyimlerini temel alarak benzer durumlar arasında bilgi transferi yapabilir ve öğrendiklerini yeni durumlara uygulayabilir. Bu, ajanın öğrenme sürecinin verimliliğini artırır.

Bu avantajlar, pekiştirmeli öğrenmeyi gerçek dünya problemleri için etkili bir öğrenme yöntemi haline getirir.

2.2.Pekiştirmeli Öğrenmenin Dezavantajları

Pekiştirmeli öğrenmenin birçok avantajının yanında bazı dezavantajlarda vardır. Bunlar:

1. Veri Verimliliği: Pekiştirmeli öğrenme genellikle veriye dayalı bir öğrenme yöntemidir. Ajanın etkileşim halinde olduğu çevrede yeterli sayıda deneyim biriktirmesi gerekebilir. Bu, bazı durumlarda zaman ve kaynak açısından maliyetli olabilir.
2. Hedef Esnekliği: Pekiştirmeli öğrenmede ajanın hedefi, toplamda alacağı ödül puanını maksimize etmek veya bir belirli hedefe ulaşmaktır. Bu, farklı problemler ve hedefler için uyarlanabilme esnekliği sağlar. Ajan, farklı hedefleri optimize etmek için gerekli stratejileri öğrenebilir.
3. Belirsizlik ve Gürültü: Pekiştirmeli öğrenme süreci, gerçek dünyadaki belirsizlik ve gürültüyle başa çıkabilme yeteneği gerektirir. Çevredeki belirsizlikler ve hatalı geri bildirimler, ajanın doğru stratejileri öğrenmesini zorlaştırabilir.
4. Hata Yapma ve Yanlış Kararlar: Pekiştirmeli öğrenme süreci, ajanın deneyimlerine dayanarak stratejisini günceller. Ancak bu süreçte hata yapma ve yanlış kararlar alma riski vardır. İlk aşamalarda yanlış seçimler yapabilen ajan, zaman içinde doğru stratejileri öğrenirken hatalardan ders alır.
5. Verimlilik ve Hız: Pekiştirmeli öğrenme süreci, bazen yavaş olabilir. Ajanın deneyimleri üzerinden sürekli öğrenmesi ve stratejisini güncellemesi zaman alabilir. Bu, bazı uygulamalar için hızlı yanıtlar gerektiren durumlarda dezavantaj olabilir.
6. Aşırı Keşif veya Aşırı Sömürü: Pekiştirmeli öğrenme sürecinde, keşif ve sömürü arasında bir denge kurmak önemlidir. Ajanın yeni fırsatları keşfetmesi ve yeni bilgiler edinmesi gerektiği gibi, aynı zamanda öğrenilen bilgilere dayanarak doğru stratejileri uygulaması da önemlidir. Dengenin sağlanması zor olabilir ve aşırı keşif veya aşırı sömürü sorunları ortaya çıkabilir.
7. Uygulama Zorlukları: Pekiştirmeli öğrenme algoritmalarının uygulanması bazen zorluklarla karşılaşabilir. İyi bir problem tanımı, uygun ödül tasarımı ve algoritmanın uygun şekilde yapılandırılması gibi faktörler dikkate alınmalıdır. Ayrıca, bazı problemler için hesaplama gücü ve veri gereksinimleri de zorluklar oluşturabilir.

Bu dezavantajlar, pekiştirmeli öğrenmenin uygulanmasını ve başarı elde etmesini zorlaştırabilen bazı faktörleri yansıtmaktadır

2.3.Değerlendirici Geri bildirim

Değerlendirici geri bildirim, pekiştirmeli öğrenme sürecinde önemli bir rol oynayan bir kavramdır. Pekiştirmeli öğrenme, bir ajanın çevresiyle etkileşime geçerek ve bu çevreden aldığı geri bildirimlere dayanarak optimal kararlar vermesini öğrenmeyi amaçlar. Bu geri bildirimler, ajanın eylemlerinin sonuçlarını değerlendirmek için kullanılır.

Değerlendirici geri bildirim, ajanın doğru kararlar için bir ödül puanı veya yanlış kararlar için ceza puanı olarak düşünülebilir. Ajan, çevredeki belirli bir durumu temsil eden bir durumda bulunur ve bu durumda bir eylem gerçekleştirir. Ardından çevreden bir geri bildirim alır. Bu geri bildirim, ajanın o durumdaki eyleminin ne kadar iyi veya kötü olduğunu gösterir. Pekiştirmeli öğrenme süreci boyunca ajanın amacı, toplamda alacağı geri bildirim miktarını maksimize etmektir. Bu amaçla ajan, geri bildirimi dikkate alarak en iyiyi seçmek için eylem stratejisini günceller ve öğrenir. Değerlendirici geri bildirim, ajanın bu öğrenme sürecinde bir yol gösterici ve hedef oluşturur. Değerlendirici geri bildirim, ajanın doğru kararlar almasını teşvik eder ve ajan, geri bildirim yardımıyla farklı durumlar ve eylemler arasındaki ilişkiyi öğrenir. Pekiştirmeli öğrenme algoritmaları, bu geri bildirimi kullanarak değer fonksiyonunu ve eylem stratejisini günceller. Böylece ajan en optimize edilmiş davranışları öğrenir.

Sonuç olarak, değerlendirici geri bildirim, pekiştirmeli öğrenme sürecinde ajanın doğru kararlar almasını sağlamak ve optimal davranışları öğrenmesi için kullanılan önemli bir bileşendir.

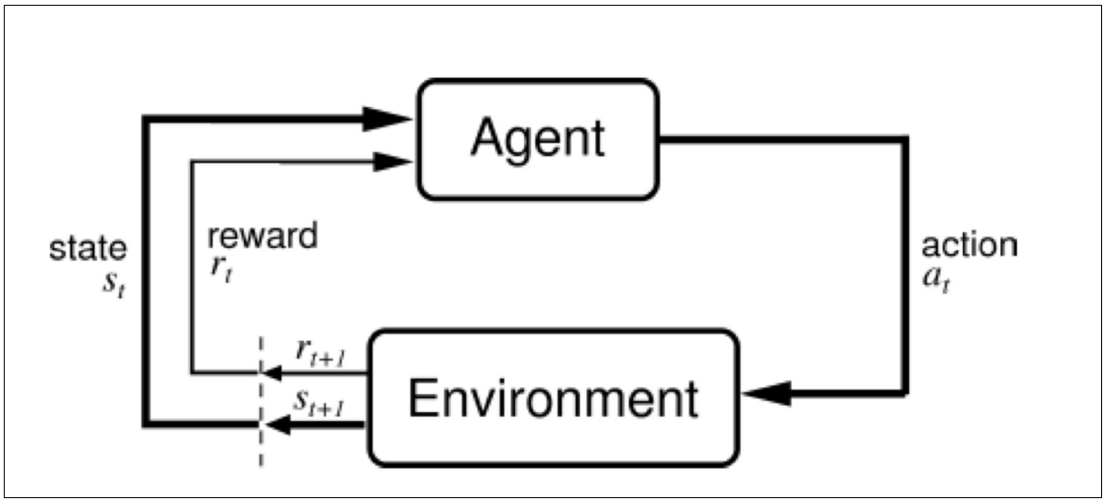
2.4.Eylem-Değer Yöntemleri

Eylem-Değer, pekiştirmeli öğrenme alanında kullanılan bir dizi algoritmadır. Bu yöntemler bir ajanın durum ve eylem arasındaki ilişkiyi öğrenerek, optimal eylem değerlerini tahmin etmesini hedefler. Pekiştirmeli öğrenmede ajan, bir durumun içinde bulunur, bir eylem gerçekleştirir ve çevreden bir geri bildirim alır. Amaç, ajanın durum-eylem çiftleri için en iyi değeri tahmin etmesini sağlamaktır. İşte burada Eylem-Değer Yöntemleri devreye girer. Eylem-Değer Yöntemleri, bir durumda belirli bir eylem gerçekleştirdiğinizde, toplam geri bildirimi (genellikle bir dizi adımlık toplam ödül olarak ifade edilen) tahmin etmek için değer fonksiyonlarını kullanır. Bu yöntemler, en iyi eylemleri seçmek için bu değer tahminlerini kullanır ve böylece ajanın optimal stratejisini belirler. En yaygın kullanılan Eylem-Değer Yöntemi algoritması, Q-öğrenmedir. Q-öğrenme, bir durumda belirli bir eylem gerçekleştirdiğinizde elde edilecek toplam geri

bildirimi tahmin etmek için bir Q değer fonksiyonunu günceller. Q-öğrenme, bu değer tahminlerini kullanarak en iyi eylemi seçer ve bu şekilde ajanın optimal stratejisini öğrenir. Diğer Eylem-Değer Yöntemi algoritmaları arasında Sarsa ve Dünya Yeterli Eylem Değerleri bulunur. Bu algoritmalar da bir ajanın durum-eylem değerlerini tahmin etmek ve optimal stratejisini belirlemek için değer fonksiyonlarını günceller. Sonuç olarak, Eylem-Değer Yöntemleri, pekiştirmeli öğrenme sürecinde ajanın durum-eylem değerlerini tahmin etmek ve optimal stratejisini belirlemek için kullanılan algoritmaları ifade eder. Bu yöntemler, değer fonksiyonlarını güncelleyerek ajanın doğru eylemleri seçmesini ve en yüksek toplam puanı elde etmesini sağlar.
(WEB_4, 2021)

2.5.Ajan-Çevre Arayüzü

Pekiştirmeli öğrenmede ajan ve çevre arasındaki etkileşim, ajan-çevre arayüzü aracılığıyla gerçekleşir. Ajan, çevreyle etkileşime geçerek durumları gözlemleyebilir, eylemler gerçekleştirebilir ve çevreden geri bildirimler alabilir. Ajan-çevre arayüzü genellikle şu şekilde işler: Durum, ajanın çevreyle etkileşime geçtiği her an, çevreden bir durum gözlemleyerek başlar. Durum, çevrenin o anki durumunu temsil eder ve ajanın kararlarını vermesi için gereken bilgiyi sağlar. Ajan, gözlemlediği duruma bağlı olarak bir eylem gerçekleştirir. Eylem, ajanın çevreye etki etme şeklidir ve farklı durumlar için farklı eylem seçenekleri olabilir. Örneğin, bir oyun oynayan bir ajanın eylemleri "ileri git", "sola dön", "ateş et" gibi olabilir. Ajan, bir eylem gerçekleştirdikten sonra çevreden bir yanıt alır. Bu yanıt, çevrenin ajanın eylemine nasıl tepki verdiğini gösterir. Yanıt, bir sonraki durumu, elde edilen ödülü veya diğer ilgili bilgileri içerebilir (Şekil 2.1). Ödül puanı, ajanın eylemlerinin değerlendirildiği bir sinyaldir. Çevre, ajanın eylemini değerlendirir ve ajanın performansına göre bir ödül veya ceza verir. Ödül, ajanın hedefini başarması durumunda pozitif bir değer alırken, hedefe ulaşmakta başarısız olması durumunda negatif bir değer alabilir. Bu ajan-çevre arayüzü, pekiştirmeli öğrenme algoritmasının çevreyle etkileşime geçmesini ve deneyimler üzerinden öğrenmesini sağlar. Ajan, durumları gözlemler, eylemler gerçekleştirir, çevreden geri bildirim alır ve bu deneyimleri kullanarak stratejisini günceller ve daha iyi kararlar almaya çalışır. Ajanın hedefi, toplamda elde edeceği ödülü maksimize etmek veya belirli bir hedefe ulaşmaktır.



Şekil 2.1. Temel pekiştirmeli öğrenme yapısı.

2.6.Ödüller ve Hedefler

Pekiştirmeli öğrenmede, ajanın amacını belirleyen ve kararlarını yönlendiren bir hedef ve bununla ilişkili olarak ödüller bulunmaktadır. Hedef, ajanın ulaşmayı amaçladığı durumu veya sonucu ifade eder. Bu hedef, genellikle önceden belirlenmiş bir amaç veya problemle ilişkilidir. Örneğin, bir oyun oynayan bir ajanın hedefi, oyunu kazanmaktır. Bir robotun hedefi, belirli bir nesneyi hedefe ulaştırmak olabilir. Hedef, ajanın stratejisini belirlemesinde önemli bir rol oynar ve pekiştirmeli öğrenme algoritmasının optimizasyon sürecinde kullanılır. Ödül, ajanın çevreden aldığı geri bildirim sinyalidir ve ajanın performansını değerlendirmek için kullanılır. Ajan, bir eylem gerçekleştirdikten sonra çevreden bir ödül alır veya cezalandırılır. Ödül, ajanın hedefe ne kadar yaklaştığını veya hedefi ne kadar başarıyla gerçekleştirdiğini yansıtan bir değerdir.

Ödül, ajanın istenen davranışı gerçekleştirmesi durumunda verilirken, ceza ise istenmeyen davranışlar için uygulanabilir. Ödüller, ajanın stratejisini geliştirmesi ve optimal eylemleri seçmesi için bir geri bildirim mekanizması olarak hizmet eder. Ödüller, ajanın durum-eylem çiftlerini değerlendirmesini sağlar ve ajan, toplamda elde edeceği ödülü maksimize etmek için stratejisini günceller. Pekiştirmeli öğrenme algoritması, ajanın hedefe ulaşmak veya toplam ödülü maksimize etmek için en iyi eylemleri seçmesini öğrenir. Ödüller, ajanın istenilen davranışlarını teşvik etmek ve yanlış davranışlardan kaçınmasını sağlamak için doğru bir şekilde tasarlanmalıdır. Ödül fonksiyonu, ajanın hedefe ulaşmakta ne kadar başarılı olduğunu ve ne zaman yanlış kararlar aldığını belirlemek için kullanılır.

3. MATCH-3 OYUNLAR

Match-3, oyuncuların benzer renkte veya desende, üç veya daha fazla nesneyi eşleştirmeye çalıştığı popüler bir oyun türüdür. Bu tür oyunlar genellikle bir oyun tahtası veya alanı üzerinde yer alan farklı renklerde veya desenlerdeki nesneleri bir araya getirmeyi hedefler. Oyuncular, nesneleri yer değiştirerek veya sıralayarak eşleştirme kombinasyonları oluşturur ve bu kombinasyonlar sayesinde puan kazanır. *Match-3* oyunları, genellikle bir hedef skor veya hamle sayısı ile gelir. Oyuncular, hedef skora mümkün olan en az hamlede ulaşmaya çalışır ve skoru maksimize etmeyi hedefler.
(WEB_5, 2020)

3.1.Motivasyon Unsurları

Hedefler ve İlerleme kullanıcılar, her bölümde belirli hedeflere ulaşmayı amaçlar. Örneğin, belirli bir puanı geçmek, belirli sayıda eşleştirme yapmak veya özel bir nesneyi tahtadan çıkarmak gibi hedefler olabilir. İlerlemelerini takip etmek ve yeni bölümlere geçmek kullanıcıları motive eder.

Zorluk ve Başarı, Oyunun sağladığı zorluk seviyeleri, kullanıcıların motivasyonunu etkileyebilir. Başarılı bir eşleştirme yapmak veya zor bir bölümü geçmek, kullanıcılarda iyi bir hissiyat ve başarı duygusu yaratır. Zorluk seviyeleri, kullanıcıları meydan okumaya teşvik edebilir ve oyunun sürekli ilgi çekici kalmasını sağlar. Kullanıcılar, bölümleri tamamladıklarında veya belirli hedeflere ulaştıklarında ödüller kazanır. Ödüller, yeni seviyelere geçme, güçlendiricilerin kilidini açma veya oyun içi kaynakları kazanma gibi avantajlar sağlayabilir. Bu ödüller, kullanıcıların ilgisini ve motivasyonunu artırır.

Sosyal etkileşim, birçok *match-3* oyunu, kullanıcıların arkadaşlarıyla veya diğer oyuncularla rekabet etmelerine veya işbirliği yapmalarına olanak sağlar. Liderlik tabloları, arkadaşlarla karşılaşma seçenekleri veya sosyal medya entegrasyonu gibi özellikler, kullanıcıların sosyal etkileşim içinde olmasını sağlar ve rekabet veya iş birliği yoluyla motivasyonu artırır.

Renkli grafikler ve görsel çekicilik*, match-3* oyunlarının renkli ve çekici grafikleri, kullanıcıları oyun dünyasına çekerek motivasyonlarını artırır. Estetik olarak hoş ve görsel açıdan zengin bir deneyim sunmak, kullanıcıların oyunu daha fazla oynamaya teşvik eder.
(WEB_6, 2012)

3.2. Bölüm Tasarımı ve Makine Öğrenimi

Match-3 oyunlarında bölüm tasarımı, kullanıcıların oyunu keyifle oynamalarını sağlamak için önemli bir faktördür. Ancak, farklı zorluk seviyelerinde dengeyi sağlamak ve kullanıcıların motivasyonunu sürdürmek zor olabilir. Neyse ki makine öğrenimi, özellikle pekiştirmeli öğrenme, bölüm tasarımının test sürecini kolaylaştırarak bu sorunu çözebilir. Pekiştirmeli öğrenme, yapay zekâ modellerini bir ortamda deney yaparak ödüllendirme ve ceza alma yoluyla en iyi stratejileri öğrenmeye teşvik eden bir öğrenme yöntemidir. *Match-3* oyunlarında bu yöntemi kullanarak kolay, orta ve zor seviyelerdeki bölümleri test etmek, geliştiricilere büyük bir avantaj sağlar.

Makine öğrenimi kullanılarak geliştirilen yapay zekâ modelleri, bölüm tasarımlarını çözerken oyunun hamle sayısını ve skorunu optimize eder. Örneğin, kolay seviye için eğitilmiş bir model, basit bölümleri başarılı bir şekilde çözebilirken, orta seviye modeli daha karmaşık bölümleri, zor seviye modeli ise en zorlu bölümleri çözebilir. Bu yapay zekâ modelleri, geliştiricilerin tasarladıkları bölümleri test etmelerine olanak sağlar. Geliştiriciler, tasarladıkları bölümü bu modeller üzerinde çalıştırarak, her seviyedeki modelin başarı oranını ve skorunu gözlemleyebilir. Böylece, tasarlanan bölümün istenilen zorluk seviyesine uygun olup olmadığını objektif bir şekilde değerlendirebilirler. Makine öğrenimi, pekiştirmeli öğrenme yöntemiyle bölüm tasarımının testini kolaylaştırarak zaman ve iş tasarrufu sağlar. Geliştiriciler, bu yöntemi kullanarak manuel olarak bölümleri test etme sürecini azaltır ve daha hızlı bir şekilde geri bildirim alır. Ayrıca, yapay zekâ modelleri sayesinde çeşitli senaryolar ve zorluk seviyeleri üzerinde deneyler yapabilir ve en iyi kullanıcı deneyimini sağlamak için tasarımı yinelemeli olarak geliştirebilirler.

Sonuç olarak, makine öğrenimi ve pekiştirmeli öğrenme kullanarak *match-3* oyunlarında bölüm tasarımı testini kolaylaştırmak, geliştiricilerin daha etkili ve dengeli bölümler oluşturmalarına yardımcı olur.

4. UYGULAMA

4.1.Geliştirme Ortamları

4.1.1. Unity 2021.3.11f1

Unity, popüler bir oyun motoru ve oyun geliştirme platformudur. Oyun motorları, oyunların tasarlanması, oluşturulması ve çalıştırılması için kullanılan yazılım araçlarıdır. Unity, oyun geliştiricilerine 2D ve 3D oyunları kolayca oluşturma ve dağıtma imkânı sağlar. Bu çalışmamızda Unity oyun motorunun 2021.3.11f1 versiyonunu kullandık. Unity oyun motorunu seçmemizdeki ana sebep pekiştirmeli öğrenmeyi desteklemesiydi.

4.1.2. C# Yazılım Dili

C#, Microsoft tarafından geliştirilen ve genellikle yazılım geliştirmede kullanılan bir programlama dilidir. C#, basit, modern ve nesne odaklı bir dil olarak bilinir. İlk olarak 2000 yılında piyasaya sürülmüş olup, Microsoft'un .NET Framework ve daha sonra .NET Core platformlarıyla birlikte yaygın olarak kullanılmaktadır. Unity oyun motoru C# yazılım dilini kullanıyor.

4.1.3. Visual Studio

Visual Studio, Microsoft tarafından geliştirilen bütünleşmiş bir geliştirme ortamıdır (IDE). Yazılım geliştirme sürecini destekleyen bir dizi araç ve özellik sunar. Visual Studio, çeşitli programlama dilleriyle çalışabilen geniş bir yelpazede uygulamalar oluşturmanıza olanak tanır. Unity oyun motoru ve C# yazılım diline verdiği destekten ötürü bu IDE'yi tercih ettik.

4.1.4. Unity ML-Agents 2.3.0

Unity ML Agents, Unity oyun motorunda yapay zekâ (AI) ve makine öğrenimi (ML) modelleri oluşturmak ve eğitmek için kullanılan bir açık kaynaklı bir kütüphanedir. "ML Agents" terimi, "Machine Learning Agents" ifadesinin kısaltmasıdır. Unity ML Agents, özellikle pekiştirmeli öğrenme algoritmalarına odaklanır. Pekiştirmeli öğrenme, bir ajanın bir ortamda belirli bir hedefi elde etmek için deney yaparak en iyi eylemleri öğrendiği bir öğrenme yöntemidir. Unity ML Agents, bu algoritmaları kullanarak oyun karakterlerini eğitmek için bir çerçeve sunar.

4.1.5. Google Sheets

Google Sheets, Google tarafından sunulan bir çevrimiçi elektronik tablo programıdır. Geleneksel masaüstü tablo uygulamalarına benzer şekilde, verileri düzenlemek, hesaplamalar yapmak ve grafikler oluşturmak için kullanılabilir. Google Sheets, Google Drive bulut depolama hizmeti ile bütünleşik çalışır ve web tarayıcısı üzerinden erişilebilir.

4.2.T-testi

T-testi, iki grup arasındaki ortalamalar arasında istatistiksel olarak anlamlı bir fark olup olmadığını test etmek için kullanılan bir hipotez testidir. İki grup arasındaki farkı belirlemek için grupların örneklem verilerine dayanır. Genellikle, bir grup ile diğer grup arasındaki farkın istatistiksel olarak anlamlı olup olmadığını belirlemek için kullanılır. T-testi, grupların normal dağılıma uyduğu ve varyansların homojen olduğu durumlarda kullanılır.

T-testi, iki temel hipotez üzerine kuruludur. Sıfır Hipotezi (H0): İki grup arasında istatistiksel olarak anlamlı bir fark yoktur. Alternatif Hipotez (H1): İki grup arasında istatistiksel olarak anlamlı bir fark vardır. T-testi, grupların örneklem verilerini kullanarak bir t değeri hesaplar. Bu t değeri, gruplar arasındaki farkın büyüklüğünü ve veri setinin değişkenliğini dikkate alır. Ardından, elde edilen t değeri, istatistiksel olarak anlamlı bir fark olup olmadığını belirlemek için bir kritik değerle karşılaştırılır. Eğer elde edilen t değeri, kritik değeri aşarsa, sıfır hipotezi reddederiz ve gruplar arasında istatistiksel olarak anlamlı bir fark olduğunu söyleriz.

Bu projeyi geliştirirken match-3 kısmı için (WEB_12, 2019) ve pekiştirmeli öğrenme kısmı için WEB_10. (2022) kaynaklarından yardım aldık. Oyun tahtasının oluşturulması ve eşleştirilebilen eşyaların kodlarını yazarken (WEB_12, 2019) kaynağını kullandık ve bazı kısımlarda değişiklikler yaptık. Pekiştirmeli öğrenmeyi projemize nasıl ekleyebiliriz kısmı için WEB_10. (2022) kaynağındaki öğreneklerden yardım aldık. Bu kaynaklardan yararlandığımız kısımlarla birlikte (WEB_13, 2022) kaynağını kullanaraktan geliştirmemizi hızlandırmaya çalıştık. Yaptığımız oyunun temel kısmını yaparken geliştirmeye açık olmasını amaçladık ve ilerleyen zamanlarda yeni özelliklerin eklenebileceğini göz önünde bulundurduk. Görsellik için (WEB_11, 2022) kaynağından yararlandık.

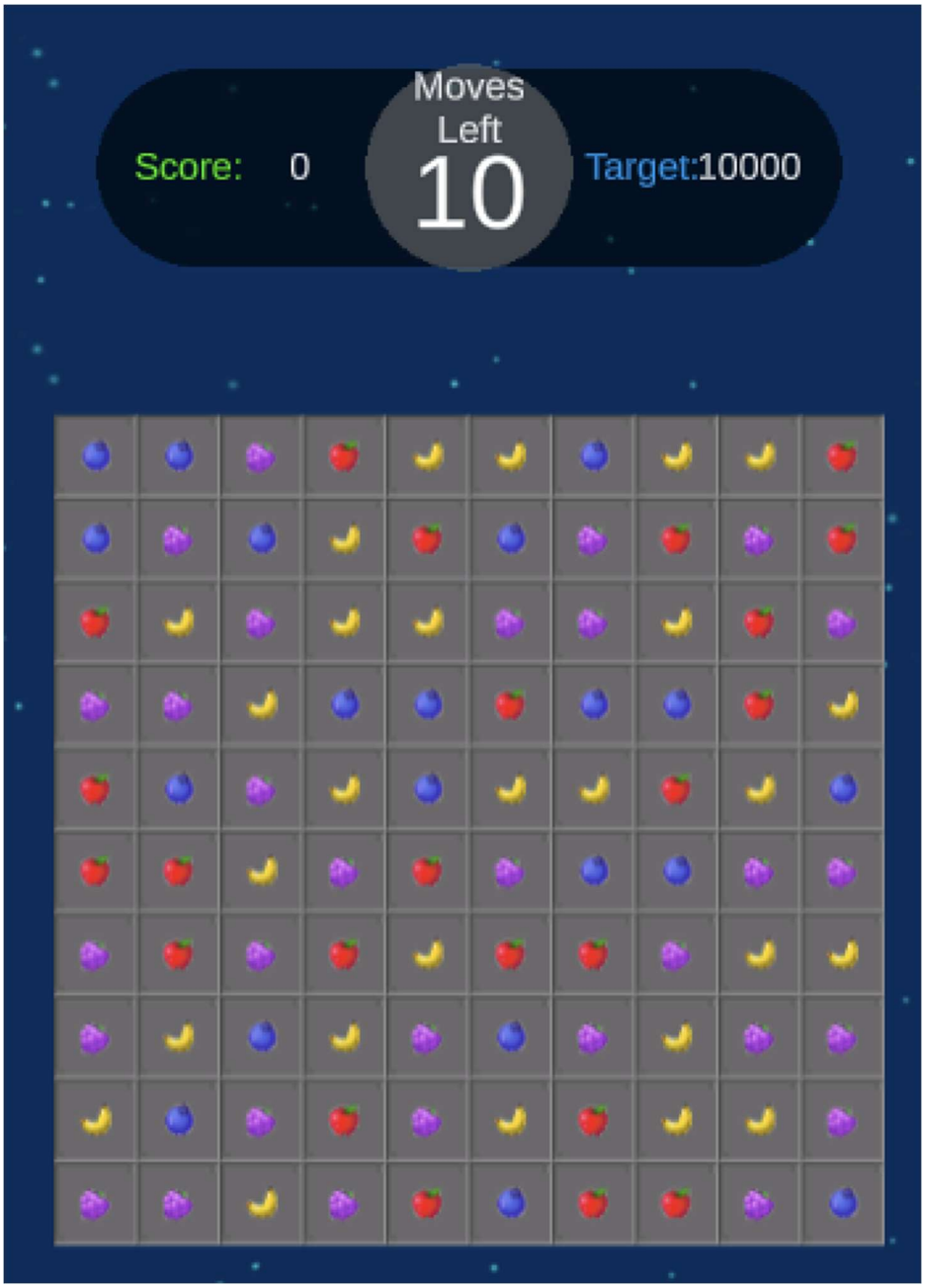
4.3.Match-3 Oyunun Geliştirilmesi

Proje kapsamında, geliştirdiğimiz oyunumuz, oyuncuların eşleştirilebilir nesneleri bir araya getirerek puan kazanabileceği ve bölümleri tamamlayabileceği bir oyun mekaniklerine sahiptir. Geliştirdiğimiz *match-3* oyunu, başka kaynaklardaki oyunlardan ilham alınarak tasarlandı. Oyunda, oyuncular ekranda yer alan nesneleri hareket ettirerek en az 3 aynı kategoride nesneyi yan yana veya üst üste getirmeye çalışır. Bu eşleştirmeler yapıldıkça, nesneler yok olur ve yerlerine yeni nesneler gelir. Oyuncular, belirli hedefleri tamamlayarak bölümleri geçmeye çalışır ve az hamle ile hedef skoru elde etmeye çalışır.

Geliştirdiğimiz *match-3* oyununun temel amacı, pekiştirmeli öğrenme yöntemini kullanarak bölüm tasarımlarını test etmek ve farklı zorluk seviyelerindeki bölümlerin oyuncular tarafından nasıl algılandığını değerlendirmektir. Oyuncuların performansı ve geri bildirimleri, makine öğrenmesi modelimizin başarısını değerlendirmek için kullandık. (WEB_9, 2020)

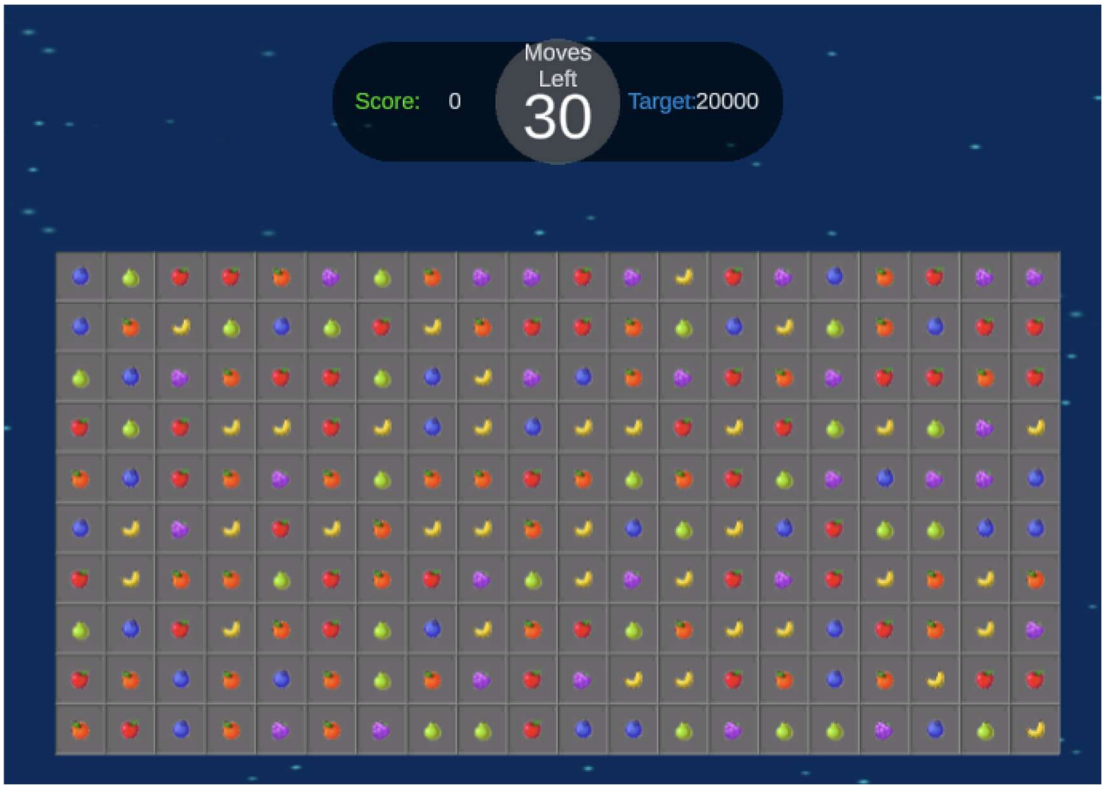
Üç farklı seviyede bölümler ile testlerimizi gerçekleştirdik:

- Kolay seviye bölüm 10 hareket sınırı ve 10000 puan hedefine sahip. (Şekil 4.1)
- Orta seviye bölüm 12 hareket 5000 puan hedefine sahip. (Şekil 4.2)
- Zor seviye bölüm 30 hareket 20000 puan hedefine sahip. (Şekil 4.3)



Şekil 4.1. Kolay seviye bölüm

Şekil 4.2. Orta seviye bölüm



Şekil 4.3. Zor seviye bölüm

4.4.Pekiştirmeli Öğrenmenin geliştirilmesi

Pekiştirmeli öğrenme yöntemlerini uygulamak için Unity ML-Agents eklentisini kullandık. Unity ML-Agents, Unity oyun motoruyla bütünleşmiş bir şekilde çalışan ve makine öğrenmesi modeli eğitimini ve uygulamasını kolaylaştıran bir araç setidir. ML-Agents kullanarak, *match-3* oyunumuz için bir ajan oluşturduk. Bu ajan, oyuncunun hareketlerini ve kararlarını taklit etmek için bir makine öğrenmesi modeli kullanarak oyunu oynamayı öğrendi. Bu öğrenme sürecinde kullandığımız puanlamalar:
(WEB_10, 2022)

- Acemi seviye model için her eşleşme 0.5 ödül puanı verirken kullanılan hareketlerde 0.25 ceza puanı verdik.
- Orta seviye model için her eşleşme 1 ödül puanı verirken kullanılan hareketlerde 0.5 ceza puanı verdik.
- İyi seviye model için her eşleşme 1 ödül puanı verirken kullanılan hareketlerde 0.5 ceza puanı verdik ve hedef hareket sayısının yarısı veya daha az hareketle bitirmişse 1 ödül puanı daha verdik.

Eğer YZ bölümü başarılı bir şekilde bitirememişse hareket sayısını bir arttırdık. Böylece inceleme yaparken son hamlede başarılı şekilde bitirenler ile bitiremeyenleri ayırmış olduk.

4.5.Modellerin Test Edilmesi

Eğitimiz modelleri t-testi ile kontrol ettik. Bu testleri yaparken 3 farklı seviyede oyuncu denek desteği aldık. Bu kişilere buradan itibaren oyuncu diyelim. Bunlardan acemi seviyede olan oyuncu, mobil oyunlar ve *match-3* oyunları oynamamış, orta seviyede olan oyuncu ise daha önceden *match-3* oynayan ama düzenli oynayamayan, iyi seviyedeki oyuncu ise gündelik olarak *match-3* oyunları oynayan bir kişiydi.

İlk bölümde acemi seviyede YZ ve oyuncu hareketlerine t-testi uyguladığımızda 0.831 elde ettik, anlamlı bir fark olması için bu sonucun 0.025'ten küçük olması gerekiyordu, yani istatistiksel olarak anlamlı bir fark çıkmadı, bu da bize ilk bölüm için acemi seviyedeki YZ'nin acemi seviyedeki oyuncuya benzediğini gösterdi (Şekil 4.4). Sonrasında orta seviye YZ ve oyuncuya ilk bölümü oynattık ve t-testinin sonucunda 0.599 elde ettik ve bu seviye içinde anlamlı bir fark olmadığı sonucunu gördük. Yani orta seviye YZ'nin orta seviye oyuncuya yakın bir sonuç verdiğini gözlemledik. İyi seviye YZ ve oyuncu ile denediğimizde de benzer bir şekilde t-testi sonucunda 0.874 elde ettik ve hareketler arasında anlamlı bir fark gözlemlemedik (Şekil 4.6).

Bu sonuçların ışığında öncelikle YZ acemi ve YZ orta, sonrasında YZ orta ve YZ iyi arasında anlamlı bir fark var mı diye t-testi uyguladık ve aralarında anlamlı bir fark tespit ettik (Şekil 4.7). Bu da bize farklı seviyelerdeki YZ modellerimizin beklendiği gibi birbirine benzemediğini gösterdi. Aynı şekilde, acemi ile orta ve orta ile iyi oyuncular arasında t-testi uyguladık. Bunun sonucunda anlamlı bir fark gözlemledik ve seviyelerin birbirinden farklı olduğunu görmüş olduk.

İlk bölümde aldığımız sonuçlardan sonra, oyunun orta seviyedeki ikinci bölümde de bu testleri gerçekleştirdik (Şekil 4.8). Acemi seviye YZ ve acemi seviye oyuncunun hareketlerini t-testine tabi tuttuğumuzda 0.051 sonucunu elde ettik ve anlamlı bir fark gözlemlemedik. Orta seviye YZ ve orta seviye oyuncu hareketlerini t-testi ile incelediğimizde 0.029 elde ettik ve 0.025'ten büyük olduğu için anlamlı bir fark gözlemlemedik. İyi seviye YZ ve iyi seviye oyuncu hareketlerini t-testi ile incelediğimizde ise 0.0923 elde ettik ve anlamlı bir fark gözlemlemedik.

Son olarak üçüncü zor seviye bölümde testlerimizi gerçekleştirdik. YZ ve oyuncu hareketleri arasında kendi seviyelerinde t-testi sonucunda anlamlı bir fark gözlemlenmedi ve

sonuç olarak YZ modellerimizin farklı zorluklardaki seviyelerde kendi seviyelerine denk gelmesi beklenen oyuncularla benzer sonuçlar elde ettiğini gözlemlemiş olduk.

	Human_Bad		ML_Bad		T-Test	Human & ML Bad
	Human_Bad_Moves	Human_Bad_Score	ML_Bad_Moves	ML_Bad_Score	Move	0.831
	4	19200	5	20100		
	4	16400	3	15800		
	5	10700	5	11200		
	9	10300	4	19100		
	3	10200	3	10700		
	5	17700	6	18600		
	5	11400	5	11700		
	7	11100	5	11200		
	7	10500	11	5700		
	11	7500	5	17600		
	11	9700	9	10300		
	11	7600	10	8200		
	5	10200	10	1120		
	3	10700	9	10100		
	8	11200	10	12700		
	8	16900	6	19300		
	6	10800	11	1010		
	7	12200	9	14100		
	4	10000	8	10500		
	4	11200	4	11600		
	11	9200	9	11000		
	8	12100	9	14000		
	10	10200	5	10500		
	5	11600	4	21200		
Aritmetik ortalama:	6.708	11608.333	6.875	12388.750		
Geometrik ortalama:	6.190	11292.243	6.351	10384.822		

Şekil 4.4. Seviye bir için acemi seviye oyuncu ve YZ

	Human_Mid		ML_Mid		T-Test	Human & ML Mid
	Human_Mid_Moves	Human_Mid_Score	ML_Mid_Moves	ML_Mid_Score	Move	0.599
	4	21700	3	22800		
	3	16000	3	22600		
	3	20100	6	12700		
	4	17000	5	16500		
	2	11300	5	11900		
	4	10100	4	10500		
	5	12800	4	11000		
	3	11000	2	10500		
	7	11500	6	10400		
	6	10200	6	10300		
	4	18700	2	22400		
	4	17000	4	13700		
	6	20500	3	10200		
	5	14900	6	16500		
	4	18400	5	10400		
	2	10400	5	13700		
	5	18700	5	25900		
	6	14200	6	11700		
	4	11600	5	10300		
	8	13100	5	10600		
	4	24500	5	24400		
	5	10200	5	11700		
	4	10700	6	10700		
	3	10000	4	11300		
Aritmetik ortalama:	4.375	14775.000	4.583	14279.167		
Geometrik ortalama:	4.140	14195.499	4.382	13524.609		

Şekil 4.5. Seviye bir için orta seviye oyuncu ve YZ

	Human_Good		ML_Good		T-Test	Human & ML Good
	Human_Good_Moves	Human_Good_Moves	ML_Good_Moves	ML_Good_Score	Move	0.874
	3	20300	2	12800		
	3	12500	4	10900		
	4	24400	4	33300		
	2	12200	3	12100		
	4	12900	4	11000		
	4	11600	4	17700		
	4	11400	2	20400		
	3	16000	4	21900		
	5	11200	3	10800		
	2	10100	4	12100		
	4	18000	4	18300		
	4	14200	2	12400		
	3	16200	3	14100		
	5	11400	4	18000		
	4	12200	5	10400		
	3	20800	3	13500		
	2	10200	4	11800		
	4	14100	3	19800		
	3	22500	4	12000		
	6	11200	4	11600		
	4	12600	4	13200		
	3	16800	4	28000		
	4	24600	3	12100		
	2	20000	4	12900		
Aritmetik ortalama:	3.542	15308.333	3.542	15462.500		
Geometrik ortalama:	3.395	14710.469	3.445	14659.051		

Şekil 4.6. Seviye bir için iyi seviye oyuncu ve YZ

T-Test	ML Bad/Mid	ML Mid/Good	Human Bad/Mid	Human Mid/Good
Move	0.0002098817887	0.0005738099506	0.0002661368236	0.01356756644

Şekil 4.7. Seviye bir için YZ ve oyuncunun kendi içlerinde hareket incelemesi

A	B	C	D	E	F	G	H	I	J	K	L	M	N
Bot		Human_Bad		Human_Mid		Human_Good		ML_Bad		ML_Mid		ML_Good	
Bot_Moves	Bot_Score	Human_Bad_Moves	Human_Bad_Score	Human_Mid_Moves	Human_Mid_Score	Human_Good_Moves	Human_Good_Moves	ML_Bad_Moves	ML_Bad_Score	ML_Mid_Moves	ML_Mid_Score	ML_Good_Moves	ML_Good_Score
4	5200	10	5100	6	5300	4	5200	10	5200	5	5100	2	10500
8	6400	11	5900	8	6800	5	5800	11	6200	12	7000	5	5100
3	6600	8	5100	7	5200	6	6800	8	5300	10	5200	4	5500
3	8700	9	5200	6	5700	3	6700	10	5200	8	5100	5	5500
5	5000	7	5000	6	5100	7	5100	7	5200	11	5000	4	5900
8	5300	7	6700	5	6200	3	5500	10	5800	6	5700	5	5700
4	7800	8	5100	9	6100	5	6700	13	4800	6	6300	3	5200
7	5500	13	4700	7	5700	4	5600	9	5200	10	5100	5	5500
6	9300	6	5800	8	5100	5	5100	8	6000	6	5500	6	6200
6	5000	8	5300	8	5700	6	5900	11	5100	10	5300	6	6500
5	8000	8	5000	5	5100	4	5500	10	5000	9	5100	5	6200
5	6600	5	5100	7	5100	8	7200	10	5100	12	5100	4	5100
4	5400	8	5100	6	5300	5	6200	8	5100	5	5200	6	6100
4	5600	6	5200	5	6800	6	5100	8	5200	10	7700	7	5100
3	5200	9	5000	6	6900	6	5800	10	5000	7	5400	7	5200
6	7100	9	5300	7	7100	5	6400	7	6200	6	7000	7	5800
5	5200	9	5100	4	7000	3	6600	13	4100	10	5100	3	5000
4	5300	6	5800	9	5100	7	5500	9	5900	8	5200	2	5000
4	5100	7	5200	7	6100	4	5700	9	5000	8	5100	6	5000
9	7100	10	5100	6	6400	6	5100	8	5500	7	5500	4	5400
4	7700	7	5200	5	6300	3	7800	12	5700	3	6700	6	6500
6	5100	13	4800	6	6700	5	5800	8	5900	3	5200	7	5000
5	5900	9	5300	7	5700	6	6000	8	5700	6	6100	3	7500
6	5400	7	5900	4	6500	3	5500	9	5700	8	5300	7	
5.167	6229.167	8.333	5291.667	6.417	5958.333	4.958	5941.667	9.417	5379.167	7.750	5625.000	4.958	5847.826
4.934	6113.697	8.109	5275.697	6.271	5918.846	4.754	5901.607	9.278	5356.935	7.286	5580.927	4.665	5758.102

Şekil 4.8. Seviye iki için değerler

T-Test	Human & ML Bad	Human & ML Mid	Human & ML Good	
Move	0.051	0.029	0.923	
T-Test	ML Bad/Mid	ML Mid/Good	Human Bad/Mid	Human Mid/Good
Move	0.005328352696	0.00001482610859	0.000200819214	0.0003929633871

Şekil 4.9. Seviye iki için sonuçlar

Bot		Human_Bad		Human_Mid		Human_Good		ML_Bad		ML_Mid		ML_Good	
Bot_Moves	Bot_Score	Human_Bad_Moves	Human_Bad_Score	Human_Mid_Moves	Human_Mid_Score	Human_Good_Moves	Human_Good_Moves	ML_Bad_Moves	ML_Bad_Score	ML_Mid_Moves	ML_Mid_Score	ML_Good_Moves	ML_Good_Score
4	5200	10	5100	6	5300	4	5200	10	5200	5	5100	2	10500
8	6400	11	5900	8	6800	5	5800	11	6200	12	7000	5	5100
3	6600	8	5100	7	5200	6	6800	8	5300	10	5200	4	5500
3	8700	9	5200	6	5700	3	6700	10	5200	8	5100	5	5500
5	5000	7	5000	6	5100	7	5100	7	5200	11	5000	4	5900
8	5300	7	6700	5	6200	3	5500	10	5800	6	5700	5	5700
4	7800	8	5100	9	6100	5	6700	13	4800	6	6300	3	5200
7	5500	13	4700	7	5700	4	5600	9	5200	10	5100	5	5500
6	9300	6	5800	8	5100	5	5100	8	6000	6	5500	6	6200
6	5000	8	5300	8	5700	6	5900	11	5100	10	5300	6	6500
5	8000	8	5000	5	5100	4	5500	10	5000	9	5100	5	6200
5	6600	5	5100	7	5100	8	7200	10	5100	12	5100	4	5100
4	5400	8	5100	6	5300	5	6200	8	5100	5	5200	6	6100
4	5600	6	5200	5	6800	6	5100	8	5200	10	7700	6	5100
3	5200	9	5000	6	6900	6	5800	10	5000	7	5400	7	5200
6	7100	9	5300	7	7100	5	6400	7	6200	6	7000	7	5800
5	5200	9	5100	4	7000	3	6600	13	4100	10	5100	3	5000
4	5300	6	5800	9	5100	7	5500	9	5900	8	5200	2	5000
4	5100	7	5200	7	6100	4	5700	9	5000	8	5100	6	5000
9	7100	10	5100	6	6400	6	5100	8	5500	7	5500	4	5400
4	7700	7	5200	5	6300	3	7800	12	5700	3	6700	6	6500
6	5100	13	4800	6	6700	5	5800	8	5900	3	5200	7	5000
5	5900	9	5300	7	5700	6	6000	8	5700	6	6100	3	7500
6	5400	7	5900	4	6500	3	5500	9	5700	8	5300	7	
5.167	6229.167	8.333	5291.667	6.417	5958.333	4.958	5941.667	9.417	5379.167	7.750	5625.000	4.917	5847.826
4.934	6113.697	8.109	5275.697	6.271	5918.846	4.754	5901.607	9.278	5356.935	7.286	5580.927	4.635	5758.102

Şekil 4.10. Seviye üç için değerler

T-Test	Human & ML Bad	Human & ML Mid	Human & ML Good	
Move	0.790	0.486	0.520	
T-Test	ML Bad/Mid	ML Mid/Good	Human Bad/Mid	Human Mid/Good
Move	0.01280003635	0.02068127547	0.00004704841512	0.000006962584307

Şekil 4.11. Seviye üç için sonuçlar

5. SONUÇ VE DEĞERLENDİRME

Çalışmamızda, bir *match-3* oyununda kolay, orta ve zor bölümleri pekiştirmeli öğrenme ile test etme ve hedeflenen düzeyde tasarlanıp tasarlanmadıklarını test etmeyi amaçladık. Üç farklı seviyede (kolay, orta ve zor) testler gerçekleştirdik ve t-testi kullanarak istatistiksel olarak seviyeler arasında anlamlı bir fark bulduk. Elde ettiğimiz istatistiksel olarak anlamlı farklar, pekiştirmeli öğrenmenin bölümlerin tasarımında etkili olduğunu göstermektedir. Sonuç ve değerlendirme olarak, pekiştirmeli öğrenme yönteminin *match-3* oyunlarında bölüm tasarımını test etmekte etkili olduğunu gösterdiğinden bahsedebiliriz.

KAYNAKÇA

Richard S. Sutton and Andrew G. Barto (1988). Reinforcement Learning: An Introduction. (1st ed.). The MIT Press Cambridge, Massachusetts London, England

WEB_1. (2019). Yapay Zekâ Araştırma İnisiyatifi web site. https://yz-ai.github.io/blog/pekistirmeli-ogrenme/pekistirmeli-ogrenme-bolum-1, 01/01/2019

WEB_2. (2001). Rice Üniversitesi Dagstuhl Semineri Exploration of Large State Spaces web site. https://www.cs.rice.edu/~vardi/dag01/givan1.pdf, 04/11/2001

WEB_3. (2020). Towards Data Science web site. https://towardsdatascience.com/intro-to-reinforcement-learning-the-explore-exploit-dilemma-463ceb004989, 25/08/2020

WEB_4. (2021). MLQ.ai web site. https://www.mlq.ai/reinforcement-learning-action-value-function/, 25/08/2020

WEB_5. (2020). AppLovin web site. https://www.applovin.com/blog/what-are-match-3-games/, 23/09/2020

WEB_6. (2012). ResearchGateweb site. https://www.researchgate.net/publication/258386136_Analysis_of_Motivational_Elements_of_Social_Games_A_Puzzle_Match_3-Games_Study_Case, 14/12/2012

WEB_7. (2019). Youtube web site. https://www.youtube.com/watch?v=DONfiKrp-QM, 17/05/2019

WEB_8. (2022). GitHub web site. https://github.com/Unity-Technologies/ml-agents, 30/11/2022

WEB_9. (2020). Youtube web site https://www.youtube.com/watch?v=DGDYU6BoVqM, 11/11/2020

WEB_10. (2022). GitHub web site. https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Learning-Environment-Examples.md,30/11/2022

WEB_11. (2022). Unity Asset Store web site. https://assetstore.unity.com/packages/2d/gui/hungry-bat-match-3-ui-free-229197, 02/09/2022

WEB_12. (2019). Youtube web site. https://www.youtube.com/watch?v=waEsGu--9P8 04/10/2019

WEB_13. (2022). Unity Asset Store web site. https://assetstore.unity.com/packages/tools/animation/dotween-hotween-v2-27676 10/10/2022

EKLER

Oyunun Kaynak Kodu

```
//Tahtayı oluşturma (WEB_12, 2019)
public class Grid<T>
{
    public event EventHandler<GridObjectChangedEventArgs> OnGridObjectChanged;
    public class GridObjectChangedEventArgs : EventArgs
    {
        public int X;
        public int Y;
    }
    private readonly int _width;
    private readonly int _height;
    private readonly float _cellSize;
    private readonly Vector3 _originPosition;
    private readonly T[,] _gridArray;
    public Grid(int width, int height, float cellSize, Vector3 originPosition, Func<Grid<T>,
int, int, T> createGridObject)
    {
        _width = width;
        _height = height;
        _cellSize = cellSize;
        _originPosition = originPosition;
        _gridArray = new T[width, height];

        for (int x = 0; x < _gridArray.GetLength(0); x++)
        {
            for (int y = 0; y < _gridArray.GetLength(1); y++)
            {
                _gridArray[x, y] = createGridObject(this, x, y);
            }
        }
    }
    public int GetWidth()
    {
        return _width;
    }
    public int GetHeight()
    {
        return _height;
    }
    public Vector3 GetWorldPosition(int x, int y)
    {
```

```
        return new Vector3(x, y) * _cellSize + _originPosition;
    }

    public void GetXY(Vector3 worldPosition, out int x, out int y)
    {
        x = Mathf.FloorToInt((worldPosition - _originPosition).x / _cellSize);
        y = Mathf.FloorToInt((worldPosition - _originPosition).y / _cellSize);
    }
    public void TriggerGridObjectChanged(int x, int y)
    {
        OnGridObjectChanged?.Invoke(this, new GridObjectChangedEventArgs { X = x, Y =
y });
    }
    public T GetGridObject(int x, int y)
    {
        if (x >= 0 && y >= 0 && x < _width && y < _height)
        {
            return _gridArray[x, y];
        }

        return default;
    }

// Yeni taş oluşturmak için
[CreateAssetMenu(fileName = "NewItem", menuName = "Item")]
public class ItemSO : ScriptableObject
{
    public Sprite sprite;
}


//Bölüm oluşturmak için (WEB_12, 2019)
[CreateAssetMenu(fileName = "NewLevel", menuName = "Level")]
public class LevelSO : ScriptableObject
{
    public enum GoalType
    {
        Score,
    }
    public List<ItemSO> gemList;
    public int width;
    public int height;
    public List<LevelGridPosition> levelGridPositionList;
    public GoalType goalType;
    public int moveAmount;
    public int targetScore;
    [System.Serializable]
    public class LevelGridPosition
```

```
    {
        public ItemSO itemSo;
        public int x;
        public int y;
        public bool hasGlass;

    }
}

// Oyunun ana yönetim kısmı (WEB_12, 2019)
public class GameManager : MonoBehaviour
{
    public event EventHandler OnGemGridPositionDestroyed;
    public event EventHandler<OnNewItemGridSpawnedEventArgs>
OnNewItemGridSpawned;
    public event EventHandler<OnLevelSetEventArgs> OnLevelSet;
    public event EventHandler OnMoveUsed;
    public event EventHandler OnOutOfMoves;
    public event EventHandler OnScoreChanged;
    public event EventHandler OnWin;
    public class OnNewItemGridSpawnedEventArgs : EventArgs
    {
        public ItemGrid ItemGrid;
        public ItemGridPosition ItemGridPosition;
    }
    public class OnLevelSetEventArgs : EventArgs
    {
        public LevelSO LevelSo;
        public Grid<ItemGridPosition> grid;
    }
    [SerializeField] private LevelSO levelSO;
    [SerializeField] private bool match4Explosions;
    private int _gridWidth;
    private int _gridHeight;
    private Grid<ItemGridPosition> _grid;
    private int _score;
    private int _moveCount;
    private void Awake()
    {
        SetLevelSo(levelSO);
    }

    public LevelSO GetLevelSO()
    {
        return levelSO;
    }
```

```
    private void SetLevelSo(LevelSO levelSO)
    {
        this.levelSO = levelSO;

        _gridWidth = levelSO.width;
        _gridHeight = levelSO.height;
        _grid = new Grid<ItemGridPosition>(_gridWidth, _gridHeight, 1f, Vector3.zero,
(Grid<ItemGridPosition> g, int x, int y) => new ItemGridPosition(g, x, y));
        for (int x = 0; x < _gridWidth; x++)
        {
            for (int y = 0; y < _gridHeight; y++)
            {
                var levelGridPosition =
levelSO.levelGridPositionList.FirstOrDefault(tmpLevelGridPosition =>
tmpLevelGridPosition.x == x && tmpLevelGridPosition.y == y);

                if (levelGridPosition == null)
                {
                    Debug.LogError("Error! Null!");
                }
                var item = levelGridPosition.itemSo;
                var itemGrid = new ItemGrid(item, x, y);
                _grid.GetGridObject(x, y).SetItemGrid(itemGrid);
                _grid.GetGridObject(x, y).SetHasGlass(levelGridPosition.hasGlass);
            }
        }
        _score = 0;
        _moveCount = levelSO.moveAmount;
        OnLevelSet?.Invoke(this, new OnLevelSetEventArgs { LevelSo = levelSO, grid =
_grid });
    }
    public int GetScore()
    {
        return _score;
    }
    public bool HasMoveAvailable()
    {
        return _moveCount > 0;
    }
    public int GetLeftMoveCount()
    {
        return _moveCount;
    }
    public int GetUsedMoveCount()
    {
        return levelSO.moveAmount - _moveCount;
    }
    public void UseMove()
```

```
    {
        _moveCount--;
        OnMoveUsed?.Invoke(this, EventArgs.Empty);
    }

    public int GetGlassAmount()
    {
        int glassAmount = 0;
        for (int x = 0; x < _gridWidth; x++)
        {
            for (int y = 0; y < _gridHeight; y++)
            {
                ItemGridPosition itemGridPosition = _grid.GetGridObject(x, y);
                if (itemGridPosition.HasGlass())
                {
                    glassAmount++;
                }
            }
        }
        return glassAmount;
    }
    public bool CanSwapGridPositions(int startX, int startY, int endX, int endY)
    {
        if (!IsValidPosition(startX, startY) || !IsValidPosition(endX, endY)) return false;

        if (startX == endX && startY == endY) return false;

        SwapGridPositions(startX, startY, endX, endY);

        bool hasLinkAfterSwap = HasMatch3Link(startX, startY) || HasMatch3Link(endX,
endY);
        SwapGridPositions(startX, startY, endX, endY);
        return hasLinkAfterSwap;
    }
    public void SwapGridPositions(int startX, int startY, int endX, int endY)
    {
        if (!IsValidPosition(startX, startY) || !IsValidPosition(endX, endY)) return;
        if (startX == endX && startY == endY) return;
        ItemGridPosition startItemGridPosition = _grid.GetGridObject(startX, startY);
        ItemGridPosition endItemGridPosition = _grid.GetGridObject(endX, endY);
        ItemGrid startItemGrid = startItemGridPosition.GetItemGrid();
        ItemGrid endItemGrid = endItemGridPosition.GetItemGrid();
        startItemGrid.SetItemXY(endX, endY);
        endItemGrid.SetItemXY(startX, startY);
        startItemGridPosition.SetItemGrid(endItemGrid);
        endItemGridPosition.SetItemGrid(startItemGrid);
    }
    public void TestingGetAllMatch3Links(int startX, int startY, int endX, int endY)
```

```
    {
        SwapGridPositions(startX, startY, endX, endY);
        List<List<ItemGridPosition>> allLinkedGemGridPositionList =
GetAllMatch3Links();

        SwapGridPositions(startX, startY, endX, endY);
    }
    public bool TryFindMatchesAndDestroyThem()
    {
        List<List<ItemGridPosition>> allLinkedGemGridPositionList =
GetAllMatch3Links();
        bool foundMatch = false;
        List<Vector2Int> explosionGridPositionList = new List<Vector2Int>();
        foreach (List<ItemGridPosition> linkedGemGridPositionList in
allLinkedGemGridPositionList)
        {
            foreach (ItemGridPosition gemGridPosition in linkedGemGridPositionList)
            {
                TryDestroyGemGridPosition(gemGridPosition);
            }
            if (linkedGemGridPositionList.Count >= 4)
            {
                _score += 200;
                ItemGridPosition explosionOriginItemGridPosition =
linkedGemGridPositionList[0];
                int explosionX = explosionOriginItemGridPosition.GetX();
                int explosionY = explosionOriginItemGridPosition.GetY();
                explosionGridPositionList.Add(new Vector2Int(explosionX - 1, explosionY -
1));
                explosionGridPositionList.Add(new Vector2Int(explosionX + 0, explosionY -
1));
                explosionGridPositionList.Add(new Vector2Int(explosionX + 1, explosionY -
1));
                explosionGridPositionList.Add(new Vector2Int(explosionX - 1, explosionY +
0));
                explosionGridPositionList.Add(new Vector2Int(explosionX + 1, explosionY +
0));
                explosionGridPositionList.Add(new Vector2Int(explosionX - 1, explosionY +
1));
                explosionGridPositionList.Add(new Vector2Int(explosionX + 0, explosionY +
1));
                explosionGridPositionList.Add(new Vector2Int(explosionX + 1, explosionY +
1));
            }

            foundMatch = true;
        }
        bool spawnExplosion = match4Explosions;
```

```
        if (spawnExplosion)
        {
            foreach (var gemGridPosition in from explosionGridPosition in
explosionGridPositionList where IsValidPosition(explosionGridPosition.x,
explosionGridPosition.y) select _grid.GetGridObject(explosionGridPosition.x,
explosionGridPosition.y))
            {
                TryDestroyGemGridPosition(gemGridPosition);
            }
        }
        OnScoreChanged?.Invoke(this, EventArgs.Empty);

        return foundMatch;
    }
    private void TryDestroyGemGridPosition(ItemGridPosition itemGridPosition)
    {
        if (itemGridPosition.HasItemGrid())
        {
            _score += 100;
            itemGridPosition.DestroyItem();
            OnGemGridPositionDestroyed?.Invoke(itemGridPosition, EventArgs.Empty);
            itemGridPosition.ClearItemGrid();
        }
    }
    public void SpawnNewMissingGridPositions()
    {
        for (int x = 0; x < _gridWidth; x++)
        {
            for (int y = 0; y < _gridHeight; y++)
            {
                ItemGridPosition itemGridPosition = _grid.GetGridObject(x, y);

                if (itemGridPosition.IsEmpty())
                {
                    var item = levelSO.itemList[UnityEngine.Random.Range(0,
levelSO.itemList.Count)];
                    var itemGrid = new ItemGrid(item, x, y);
                    itemGridPosition.SetItemGrid(itemGrid);
                    OnNewItemGridSpawned?.Invoke(itemGrid, new
OnNewItemGridSpawnedEventArgs
                    {
                        ItemGrid = itemGrid,
                        ItemGridPosition = itemGridPosition,
                    });
                }
            }
        }
    }
```

```
public void FallItemsIntoEmptyPositions()
{
   for (int x = 0; x < _gridWidth; x++)
   {
      for (int y = 0; y < _gridHeight; y++)
      {
         ItemGridPosition itemGridPosition = _grid.GetGridObject(x, y);

         if (!itemGridPosition.IsEmpty())
         {
            for (int i = y - 1; i >= 0; i--)
            {
               ItemGridPosition nextItemGridPosition = _grid.GetGridObject(x, i);
               if (nextItemGridPosition.IsEmpty())
               {
                  itemGridPosition.GetItemGrid().SetItemXY(x, i);
                  nextItemGridPosition.SetItemGrid(itemGridPosition.GetItemGrid());
                  itemGridPosition.ClearItemGrid();
                  itemGridPosition = nextItemGridPosition;
               }
               else
               {
                  break;
               }
            }
         }
      }
   }
}

private bool HasMatch3Link(int x, int y)
{
   var links = GetMatch3Links(x, y);
   return links != null && links.Count >= 3;
}
private List<ItemGridPosition> GetMatch3Links(int x, int y)
{
   var itemSo = GetItemSo(x, y);
   if (itemSo == null) return null;
   int rightLinkAmount = 0;
   for (int i = 1; i < _gridWidth; i++)
   {
      if (IsValidPosition(x + i, y))
      {
         var nextItemSo = GetItemSo(x + i, y);
         if (nextItemSo == itemSo)
         {
            rightLinkAmount++;
```

```
        }
        else
        {
            break;
        }
    }
    else
    {
        break;
    }
}
int leftLinkAmount = 0;
for (int i = 1; i < _gridWidth; i++)
{
    if (IsValidPosition(x - i, y))
    {
        var nextItemSo = GetItemSo(x - i, y);
        if (nextItemSo == itemSo)
        {
            leftLinkAmount++;
        } else
        {
            break;
        }
    }
    else
    {
        break;
    }
}
int horizontalLinkAmount = 1 + leftLinkAmount + rightLinkAmount;
if (horizontalLinkAmount >= 3)
{
    var list = new List<ItemGridPosition>();
    int leftMostX = x - leftLinkAmount;
    for (int i = 0; i < horizontalLinkAmount; i++)
    {
        list.Add(_grid.GetGridObject(leftMostX + i, y));
    }
    return list;
}
int upLinkAmount = 0;
for (int i = 1; i < _gridHeight; i++)
{
    if (IsValidPosition(x, y + i))
    {
        var nextItemSo = GetItemSo(x, y + i);
        if (nextItemSo == itemSo)
```

```
            {
                upLinkAmount++;
            }
            else
            {
                break;
            }
        }
        else
        {
            break;
        }
    }
    int downLinkAmount = 0;
    for (int i = 1; i < _gridHeight; i++)
    {
        if (IsValidPosition(x, y - i))
        {
            var nextItemSo = GetItemSo(x, y - i);
            if (nextItemSo == itemSo)
            {
                downLinkAmount++;
            }
            else
            {
                break;
            }
        }
        else
        {
            break;
        }
    }
    int verticalLinkAmount = 1 + downLinkAmount + upLinkAmount;
    if (verticalLinkAmount >= 3)
    {
        var list = new List<ItemGridPosition>();
        int downMostY = y - downLinkAmount;
        for (int i = 0; i < verticalLinkAmount; i++)
        {
            list.Add(_grid.GetGridObject(x, downMostY + i));
        }
        return list;
    }

    return null;
}
private List<List<ItemGridPosition>> GetAllMatch3Links()
```

```
{
   var gridPositionList = new List<List<ItemGridPosition>>();
   for (int x = 0; x < _gridWidth; x++)
   {
      for (int y = 0; y < _gridHeight; y++)
      {
         if (HasMatch3Link(x, y))
         {
            var links = GetMatch3Links(x, y);

            if (gridPositionList.Count == 0)
            {
               gridPositionList.Add(links);
            }
            else
            {
               bool uniqueNewLink = true;

               foreach (List<ItemGridPosition> itemGridPositions in gridPositionList)
               {
                  if (links.Count == itemGridPositions.Count)
                  {
                     bool allTheSame = true;
                     for (int i = 0; i < links.Count; i++)
                     {
                        if (links[i] == itemGridPositions[i])
                        {
                        }
                        else
                        {
                           allTheSame = false;
                           break;
                        }
                     }
                     if (allTheSame)
                     {
                        uniqueNewLink = false;
                     }
                  }
               }
               if (uniqueNewLink) {
                  gridPositionList.Add(links);
               }
            }
         }
      }
   }
   return gridPositionList;
```

```
    }
    public List<PossibleMove> GetAllPossibleMoves()
    {
        var allPossibleMovesList = new List<PossibleMove>();

        for (int y = 0; y < _gridHeight; y++)
        {
            for (int x = 0; x < _gridWidth; x++)
            {
                var testPossibleMoveList = new List<PossibleMove>();
                testPossibleMoveList.Add(new PossibleMove(x, y, x - 1, y + 0));
                testPossibleMoveList.Add(new PossibleMove(x, y, x + 1, y + 0));
                testPossibleMoveList.Add(new PossibleMove(x, y, x + 0, y + 1));
                testPossibleMoveList.Add(new PossibleMove(x, y, x + 0, y - 1));
                foreach (var possibleMove in testPossibleMoveList)
                {
                    bool skipPossibleMove = false;
                    foreach (var tmpPossibleMove in allPossibleMovesList)
                    {
                        if (tmpPossibleMove.startX == possibleMove.startX &&
                            tmpPossibleMove.startY == possibleMove.startY &&
                            tmpPossibleMove.endX == possibleMove.endX &&
                            tmpPossibleMove.endY == possibleMove.endY)
                        {
                            skipPossibleMove = true;
                            break;
                        }
                        if (tmpPossibleMove.startX == possibleMove.endX &&
                            tmpPossibleMove.startY == possibleMove.endY &&
                            tmpPossibleMove.endX == possibleMove.startX &&
                            tmpPossibleMove.endY == possibleMove.startY)
                        {
                            skipPossibleMove = true;
                            break;
                        }
                    }

                    if (skipPossibleMove)
                    {
                        continue;
                    }
                    SwapGridPositions(possibleMove.startX, possibleMove.startY,
possibleMove.endX, possibleMove.endY);
                    var match3Links = GetAllMatch3Links();
                    if (match3Links.Count > 0)
                    {
                        possibleMove.allLinkedItemGridPositionList = match3Links;
                        allPossibleMovesList.Add(possibleMove);
```

```
                }

                SwapGridPositions(possibleMove.startX, possibleMove.startY,
possibleMove.endX, possibleMove.endY);
            }
        }
    }
    return allPossibleMovesList;
}
public ItemSO GetItemSo(int x, int y)
{
    if (!IsValidPosition(x, y)) return null;
    var gridPosition = _grid.GetGridObject(x, y);
    return gridPosition.GetItemGrid() == null ? null :
gridPosition.GetItemGrid().GetItem();
}
private bool IsValidPosition(int x, int y)
{
    return x >= 0 && y >= 0 &&
        x < _gridWidth && y < _gridHeight;
}
public bool TryIsGameOver()
{
    if (!HasMoveAvailable())
    {
        OnOutOfMoves?.Invoke(this, EventArgs.Empty);
        return true;
    }
    switch (levelSO.goalType)
    {
        default:
        case LevelSO.GoalType.Score:
            if (_score >= levelSO.targetScore)
            {
                OnWin?.Invoke(this, EventArgs.Empty);
                return true;
            }
            break;
    }
    return false;
}
public class PossibleMove
{
    public int startX;
    public int startY;
    public int endX;
    public int endY;
    public List<List<ItemGridPosition>> allLinkedItemGridPositionList;
```

```
        public PossibleMove(int startX, int startY, int endX, int endY)
        {
            this.startX = startX;
            this.startY = startY;
            this.endX = endX;
            this.endY = endY;
        }
        public int GetTotalMatchAmount()
        {
            return allLinkedItemGridPositionList.Sum(gridPositionList =>
gridPositionList.Count);
        }
        public override string ToString()
        {
            return startX + ", " + startY + " => " + endX + ", " + endY + " == " +
allLinkedItemGridPositionList?.Count;
        }
    }
    public class ItemGridPosition
    {
        public event EventHandler OnGlassDestroyed;
        private ItemGrid _itemGrid;
        private Grid<ItemGridPosition> grid;
        private int x;
        private int y;
        private bool hasGlass;
        public ItemGridPosition(Grid<ItemGridPosition> grid, int x, int y)
        {
            this.grid = grid;
            this.x = x;
            this.y = y;
        }
        public void SetItemGrid(ItemGrid itemGrid)
        {
            _itemGrid = itemGrid;
            grid.TriggerGridObjectChanged(x, y);
        }
        public int GetX()
        {
            return x;
        }
        public int GetY()
        {
            return y;
        }
        public Vector3 GetWorldPosition()
        {
            return grid.GetWorldPosition(x, y);
```

```
    }
    public ItemGrid GetItemGrid()
    {
        return _itemGrid;
    }
    public void ClearItemGrid()
    {
        _itemGrid = null;
    }
    public void DestroyItem()
    {
        _itemGrid?.Destroy();
        grid.TriggerGridObjectChanged(x, y);
    }
    public bool HasItemGrid()
    {
        return _itemGrid != null;
    }
    public bool IsEmpty() {
        return _itemGrid == null;
    }
    public bool HasGlass()
    {
        return hasGlass;
    }
    public void SetHasGlass(bool hasGlass)
    {
        this.hasGlass = hasGlass;
    }
    public override string ToString()
    {
        var empty = string.Empty;
        return _itemGrid?.ToString() ?? empty;
    }
}
public class ItemGrid
{
    public event EventHandler OnDestroyed;
    private ItemSO _item;
    private int x;
    private int y;
    private bool isDestroyed;
    public ItemGrid(ItemSO item, int x, int y)
    {
        _item = item;
        this.x = x;
        this.y = y;
```

```
            isDestroyed = false;
        }
        public ItemSO GetItem()
        {
            return _item;
        }
        public Vector3 GetWorldPosition()
        {
            return new Vector3(x, y);
        }
        public void SetItemXY(int x, int y)
        {
            this.x = x;
            this.y = y;
        }
        public void Destroy()
        {
            isDestroyed = true;
            OnDestroyed?.Invoke(this, EventArgs.Empty);
        }
        public override string ToString()
        {
            return isDestroyed.ToString();
        }
    }
}
```

// Eşlenebilir eşyaların özellikleri (WEB_12, 2019)

```
public class Match3Visual : MonoBehaviour
{
    public event EventHandler OnStateChanged;
    public event EventHandler OnStateWaitingForUser;
    public enum State
    {
        Busy,
        WaitingForUser,
        TryFindMatches,
        GameOver,
    }
    [SerializeField] private Transform pfGemGridVisual;
    [SerializeField] private Transform pfGlassGridVisual;
    [SerializeField] private Transform pfBackgroundGridVisual;
    [SerializeField] private Transform cameraTransform;
    [SerializeField] private GameManager gameManager;
    private Grid<GameManager.ItemGridPosition> grid;
    private Dictionary<GameManager.ItemGrid, GemGridVisual> gemGridDictionary;
    private bool _isSetup;
    private State _state;
```

```
    private float _busyTimer;
    private Action _onBusyTimerElapsedAction;
    private int _startDragX;
    private int _startDragY;

    private void Awake()
    {
        _state = State.Busy;
        _isSetup = false;
        gameManager.OnLevelSet += GameManagerOnLevelSet;
    }
    private void GameManagerOnLevelSet(object sender,
GameManager.OnLevelSetEventArgs e)
    {
        FunctionTimer.Create(() => {
            Setup(sender as GameManager, e.grid);
        }, .1f);
    }
    private void Setup(GameManager gameManager,
Grid<GameManager.ItemGridPosition> grid)
    {
        this.gameManager = gameManager;
        this.grid = grid;
        float cameraYOffset = 1f;
        cameraTransform.position = new Vector3(grid.GetWidth() * .5f, grid.GetHeight() *
.5f + cameraYOffset, cameraTransform.position.z);
        gameManager.OnGemGridPositionDestroyed +=
Match3_OnGemGridPositionDestroyed;
        gameManager.OnNewItemGridSpawned += Match3OnNewItemGridSpawned;
        gemGridDictionary = new Dictionary<GameManager.ItemGrid, GemGridVisual>();
        for (int x = 0; x < grid.GetWidth(); x++)
        {
            for (int y = 0; y < grid.GetHeight(); y++)
            {
                GameManager.ItemGridPosition itemGridPosition = grid.GetGridObject(x, y);
                GameManager.ItemGrid itemGrid = itemGridPosition.GetItemGrid();
                Vector3 position = grid.GetWorldPosition(x, y);
                position = new Vector3(position.x, 12);
                Transform gemGridVisualTransform = Instantiate(pfGemGridVisual, position,
Quaternion.identity);
                gemGridVisualTransform.Find("sprite").GetComponent<SpriteRenderer>().sprite
= itemGrid.GetItem().sprite;
                GemGridVisual gemGridVisual = new
GemGridVisual(gemGridVisualTransform, itemGrid);
                gemGridDictionary[itemGrid] = gemGridVisual;
                Transform glassGridVisualTransform = Instantiate(pfGlassGridVisual,
grid.GetWorldPosition(x, y), Quaternion.identity);
                Instantiate(pfBackgroundGridVisual, grid.GetWorldPosition(x, y),
```

```
Quaternion.identity);
            }
        }

        SetBusyState(.5f, () => SetState(State.TryFindMatches));
        _isSetup = true;
    }
    private void Match3OnNewItemGridSpawned(object sender,
GameManager.OnNewItemGridSpawnedEventArgs e)
    {
        Vector3 position = e.ItemGrid.GetWorldPosition();
        position = new Vector3(position.x, 12);
        Transform gemGridVisualTransform = Instantiate(pfGemGridVisual, position,
Quaternion.identity);
        gemGridVisualTransform.Find("sprite").GetComponent<SpriteRenderer>().sprite =
e.ItemGrid.GetItem().sprite;
        GemGridVisual gemGridVisual = new GemGridVisual(gemGridVisualTransform,
e.ItemGrid);
        gemGridDictionary[e.ItemGrid] = gemGridVisual;
    }
    private void Match3_OnGemGridPositionDestroyed(object sender, EventArgs e)
    {
        if (sender is GameManager.ItemGridPosition itemGridPosition &&
itemGridPosition.GetItemGrid() != null) {
            gemGridDictionary.Remove(itemGridPosition.GetItemGrid());
        }
    }
    private void Update()
    {
        if (!_isSetup) return;
        UpdateVisual();
        switch (_state)
        {
            case State.Busy:
                _busyTimer -= Time.deltaTime;
                if (_busyTimer <= 0f)
                {
                    _onBusyTimerElapsedAction();
                }
                break;
            case State.WaitingForUser:
                if (Input.GetMouseButtonDown(0))
                {
                    var mouseWorldPosition = UtilsClass.GetMouseWorldPosition();
                    grid.GetXY(mouseWorldPosition, out _startDragX, out _startDragY);
                }

                if (Input.GetMouseButtonUp(0))
```

```
        {
            Vector3 mouseWorldPosition = UtilsClass.GetMouseWorldPosition();
            grid.GetXY(mouseWorldPosition, out int x, out int y);

            if (x != _startDragX)
            {
                y = _startDragY;

                if (x < _startDragX)
                {
                    x = _startDragX - 1;
                }
                else
                {
                    x = _startDragX + 1;
                }
            }
            else
            {
                x = _startDragX;

                if (y < _startDragY)
                {
                    y = _startDragY - 1;
                }
                else
                {
                    y = _startDragY + 1;
                }
            }
            if (gameManager.CanSwapGridPositions(_startDragX, _startDragY, x, y)) {
                SwapGridPositions(_startDragX, _startDragY, x, y);
            }
        }
        break;
    case State.TryFindMatches:
        if (gameManager.TryFindMatchesAndDestroyThem())
        {
            SetBusyState(.3f, () =>
            {
                gameManager.FallItemsIntoEmptyPositions();

                SetBusyState(.3f, () =>
                {
                    gameManager.SpawnNewMissingGridPositions();

                    SetBusyState(.5f, () => SetState(State.TryFindMatches));
                });
```

```
            });
        }
        else
        {
            TrySetStateWaitingForUser();
        }
        break;
    case State.GameOver:
        break;
  }
}
private void UpdateVisual()
{
  foreach (GameManager.ItemGrid gemGrid in gemGridDictionary.Keys)
  {
    gemGridDictionary[gemGrid].Update();
  }
}
public void SwapGridPositions(int startX, int startY, int endX, int endY)
{
  gameManager.SwapGridPositions(startX, startY, endX, endY);
  gameManager.UseMove();

  SetBusyState(.5f, () => SetState(State.TryFindMatches));
}
private void SetBusyState(float busyTimer, Action onBusyTimerElapsedAction)
{
  SetState(State.Busy);
  _busyTimer = busyTimer;
  _onBusyTimerElapsedAction = onBusyTimerElapsedAction;
}
private void TrySetStateWaitingForUser()
{
  if (gameManager.TryIsGameOver())
  {
    SetState(State.GameOver);
  }
  else
  {
    SetState(State.WaitingForUser);
    OnStateWaitingForUser?.Invoke(this, EventArgs.Empty);
  }
}
private void SetState(State state)
{
  _state = state;
  OnStateChanged?.Invoke(this, EventArgs.Empty);
}
```

```
    public State GetState()
    {
        return _state;
    }

    private class GemGridVisual
    {
        private readonly Transform _transform;
        private readonly GameManager.ItemGrid _itemGrid;

        public GemGridVisual(Transform transform, GameManager.ItemGrid itemGrid)
        {
            _transform = transform;
            _itemGrid = itemGrid;

            itemGrid.OnDestroyed += GemGrid_OnDestroyed;
        }
        private void GemGrid_OnDestroyed(object sender, EventArgs e)
        {
            _transform.GetComponent<Animation>().Play();
            Destroy(_transform.gameObject, 1f);
        }
        public void Update()
        {
            Vector3 targetPosition = _itemGrid.GetWorldPosition();
            var position = _transform.position;
            Vector3 moveDir = (targetPosition - position);
            float moveSpeed = 10f;
            position += moveDir * moveSpeed * Time.deltaTime;
            _transform.position = position;
        }
    }
}

// Oyun tahtası (WEB_10, 2022)
public class Match3MLAgentsBoard : AbstractBoard
{
    [SerializeField] private ModelMode modelMode;
    [SerializeField] private GameManager gameManager;
    [SerializeField] private Match3Visual match3Visual;
    [SerializeField] private bool autoReloadscene;
    private LevelSO levelSo;
    private Agent agent;
    private void Awake()
    {
        agent = GetComponent<Agent>();
        match3Visual.OnStateWaitingForUser += OnStateWaitingForUser;
        gameManager.OnGemGridPositionDestroyed += OnGemGridPositionDestroyed;
```

```
        gameManager.OnMoveUsed += OnMoveUsed;
        gameManager.OnOutOfMoves += EndLevel;
        gameManager.OnWin += EndLevel;
    }

    private void OnDisable()
    {
        match3Visual.OnStateWaitingForUser -= OnStateWaitingForUser;
        gameManager.OnGemGridPositionDestroyed -= OnGemGridPositionDestroyed;
        gameManager.OnMoveUsed -= OnMoveUsed;
        gameManager.OnOutOfMoves -= EndLevel;
        gameManager.OnWin -= EndLevel;
    }
    public override BoardSize GetMaxBoardSize()
    {
        levelSo = gameManager.GetLevelSO();
        var boardSize = new BoardSize();
        boardSize.Columns = levelSo.width;
        boardSize.Rows = levelSo.height;
        boardSize.NumCellTypes = levelSo.itemList.Count;
        boardSize.NumSpecialTypes = levelSo.goalType == LevelSO.GoalType.Score ? 0 :
1;
        return boardSize;
    }
    public override int GetCellType(int row, int col)
    {
        ItemSO itemSo = gameManager.GetItemSo(col, row);
        return levelSo.itemList.IndexOf(itemSo);
    }
    public override int GetSpecialType(int row, int col)
    {
        return 0;
    }
    public override bool IsMoveValid(Move m)
    {
        int startX = m.Column;
        int startY = m.Row;
        var moveEnd = m.OtherCell();
        int endX = moveEnd.Column;
        int endY = moveEnd.Row;
        return gameManager.CanSwapGridPositions(startX, startY, endX, endY);
    }
    public override bool MakeMove(Move m)
    {
        int startX = m.Column;
        int startY = m.Row;
        var moveEnd = m.OtherCell();
        int endX = moveEnd.Column;
```

```
        int endY = moveEnd.Row;
        if (gameManager.CanSwapGridPositions(startX, startY, endX, endY))
        {
            match3Visual.SwapGridPositions(startX, startY, endX, endY);
            return true;
        }
        return false;
    }
    private void OnStateWaitingForUser(object sender, System.EventArgs e)
    {
        agent.RequestDecision();
    }
    private void OnGemGridPositionDestroyed(object sender, System.EventArgs e)
    {
        if(levelSo.goalType == LevelSO.GoalType.Score)
            agent.AddReward(modelMode.MatchReward);
    }
    private void OnMoveUsed(object sender, System.EventArgs e)
    {
        agent.AddReward(modelMode.MoveReward);
    }
    private void EndLevel(object sender, System.EventArgs e)
    {
        agent.EndEpisode();
        if(autoReloadscene)
            UnityEngine.SceneManagement.SceneManager.LoadScene(sceneBuildIndex: 0);
    }
}

// YZ modelin puanlaması

[CreateAssetMenu(fileName = "NewMode", menuName = "Mode")]
public class ModelMode : ScriptableObject
{
    public float MatchReward;
    public float MoveReward;
    public float ExtraReward;
}

// Oyundaki canvas elementleri kontrolü
public class UIManager : MonoBehaviour {

    [SerializeField] private GameManager gameManager;
    public TextMeshProUGUI movesText;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI targetScoreText;
    public Transform winLoseTransform;
    public TextMeshProUGUI winLoseTransformText;
```

```
    private void Awake()
    {
        winLoseTransform.gameObject.SetActive(false);
        gameManager.OnLevelSet += GameManagerOnLevelSet;
        gameManager.OnMoveUsed += GameManagerOnMoveUsed;
        gameManager.OnScoreChanged += GameManagerOnScoreChanged;
        gameManager.OnOutOfMoves += GameManagerOnOutOfMoves;
        gameManager.OnWin += GameManagerOnWin;
    }
    private void GameManagerOnWin(object sender, System.EventArgs e)
    {
        winLoseTransform.gameObject.SetActive(true);
        winLoseTransformText.text = "<color=green>YOU WIN!</color>";
    }
    private void GameManagerOnOutOfMoves(object sender, System.EventArgs e)
    {
        winLoseTransform.gameObject.SetActive(true);
        winLoseTransformText.text = "<color=red>YOU LOSE!</color>";
    }
    private void GameManagerOnScoreChanged(object sender, System.EventArgs e)
    {
        UpdateText();
    }
    private void GameManagerOnGlassDestroyed(object sender, System.EventArgs e)
    {
        UpdateText();
    }
    private void GameManagerOnMoveUsed(object sender, System.EventArgs e)
    {
        UpdateText();
    }
    private void GameManagerOnLevelSet(object sender, System.EventArgs e)
    {
        LevelSO levelSO = gameManager.GetLevelSO();
        targetScoreText.text = levelSO.targetScore.ToString();
        UpdateText();
    }
    private void UpdateText()
    {
        movesText.text = gameManager.GetLeftMoveCount().ToString();
        scoreText.text = gameManager.GetScore().ToString();
    }
}

// Oyunun değerlerini ayarlayan kdo aprçası
public sealed class ApplicationManager : Monobehaviour
{
    public void Awake()
```

```
    {
        Application.lowMemory += MemoryCleanup;
        SceneHandler.s_CurrentAsyncOp.completed += MemoryCleanup;

        SetFramerate();
        SetDoTweenSettings();
        Screen.sleepTimeout = SleepTimeout.NeverSleep;
    }
    public override void Deinitialize()
    {
        Application.lowMemory  -= MemoryCleanup;
        SceneHandler.s_CurrentAsyncOp.completed -= MemoryCleanup;
    }
    static void MemoryCleanup()
    {
        Resources.UnloadUnusedAssets();
        GC.Collect();
    }
    static void MemoryCleanup(AsyncOperation obj) => MemoryCleanup();
    public static bool HasInternet()
    {
        return Application.internetReachability != NetworkReachability.NotReachable;
    }
    public static bool HasInternet(out bool hasInternet)
    {
        hasInternet = Application.internetReachability !=
NetworkReachability.NotReachable;
        return hasInternet;
    }
    static void SetFramerate()
    {
        if (SystemInfo.processorCount == 1 || Screen.width <= 600)
        {
            Application.targetFrameRate = 30;
        }
        else
        {
            Application.targetFrameRate = 60;
        }
    }
        private static void SetDoTweenSettings()
        {
            DOTween.Init();
            DOTween.defaultAutoPlay = AutoPlay.All;
            DOTween.defaultAutoKill = true;
```

```
        DOTween.SetTweensCapacity(250, 100);
        PLogger.Log("DoTween settings adjusted");
    }
}

//Scene'leri kontrol eden kod parçası

public sealed class SceneHandler : MonobeBaviour
  {
    public static   AsyncOperation s_CurrentAsyncOp;
    public static   string        s_CurrentSceneName =>
SceneManager.GetActiveScene().name;
    [SerializeField] SceneField    defaultScene;
    SceneField          loadingScene;
    public static SceneHandler instance;
    public void Awake()
    {
      if (instance == null)
        instance = this;
      else
        Destroy(gameObject);
      if (s_CurrentAsyncOp == null)
        StartCoroutine(AsyncLoad(defaultScene));
      StartCoroutine(Wait());
      IEnumerator Wait()
      {
        yield return new WaitUntil(() => s_CurrentAsyncOp != null);
        s_CurrentAsyncOp.completed += LoadCompleted;
      }
    }
    public override void Deinitialize()
    {
      s_CurrentAsyncOp.completed -= LoadCompleted;
    }
    public void LoadScene(SceneField sceneField, bool disableChecks = false)
    {
      string sceneIndex = sceneField;
      if (disableChecks == false)
        if (!CanLoadScene(sceneIndex))
          return;
      var x = StartCoroutine(AsyncUnload());
      s_CurrentAsyncOp  = SceneManager.LoadSceneAsync(sceneIndex,
LoadSceneMode.Additive);
      s_CurrentAsyncOp.allowSceneActivation = true;
    }
```

```
        public void LoadScene(string sceneName, bool disableChecks = false)
    {
        s_CurrentAsyncOp = SceneManager.LoadSceneAsync(sceneName,
LoadSceneMode.Additive);
        s_CurrentAsyncOp.allowSceneActivation = true;
    }
    public void UnloadThenLoad(SceneField sceneField)
    {
        StartCoroutine(AsyncUnload());
        loadingScene = sceneField;
        StartCoroutine(WaitUnloadToLoad(sceneField));
    }
    IEnumerator AsyncUnload()
    {
        s_CurrentAsyncOp = SceneManager.UnloadSceneAsync(s_CurrentSceneName);
        yield return (s_CurrentAsyncOp.progress > 0.99f);
    }
    IEnumerator AsyncLoad(SceneField sceneField)
    {
        s_CurrentAsyncOp = SceneManager.LoadSceneAsync(sceneField,
LoadSceneMode.Additive);
        s_CurrentAsyncOp.allowSceneActivation = true;
        while (!s_CurrentAsyncOp.isDone) yield return null;
        SceneManager.SetActiveScene(SceneManager.GetSceneByName(sceneField));
    }
    IEnumerator WaitUnloadToLoad(SceneField sceneField)
    {
        yield return new WaitUntil(() => s_CurrentAsyncOp.isDone);
        StartCoroutine(AsyncLoad(sceneField));
    }
    static bool CanLoadScene(string sceneName)
    {
        if (!s_CurrentAsyncOp.isDone)
        {
            Debug.LogWarning("A scene already being loaded, operation skipped.");
            return false;
        }
        if (sceneName == s_CurrentSceneName)
        {
            Debug.LogWarning("Same scene passed, skipping.");
            return false;
        }
        return true;
    }
```

```
    static bool CanLoadScene(string sceneName, out bool canLoad)
    {
        if (!s_CurrentAsyncOp.isDone)
        {
            Debug.LogWarning("A scene already being loaded, operation skipped.");
            return canLoad = false;
        }
        if (sceneName == s_CurrentSceneName)
        {
            Debug.LogWarning("Same scene passed, skipping.");
            return canLoad = false;
        }
        return canLoad = true;
    }
}
```

//Sesleri kontrol eden kod parçası

```
public class AudioManager : MonoBehaviour
{
    [SerializeField] Audios audios;
    [SerializeField] AudioSource sfxStandartSource;
    [SerializeField] AudioSource sfxIncreasingSource;
    [SerializeField] AudioSource musicSource;
    AudioModel currentMusic;
    static float audioVolume;
    static bool  isAudioEnabled = false;
    public void Awake()
    {
        base.Init();
        SetAudioState(SettingsData.SfxData);
    }
    public void PlayMusic(string name, bool fadeOut = false, float fadeTime = 1f)
    {
        var music = audios.m_Musics.FirstOrDefault(x => x.name == name);
        PlayMusic(music, fadeOut, fadeTime);
    }
    public void PlayMusic(int id, bool fadeOut = false, float fadeTime = 1f)
    {
        var music = audios.m_Musics.FirstOrDefault(x => x.id == id);
        PlayMusic(music, fadeOut, fadeTime);
    }
    public void PlaySFX(string name)
    {
        var sfx = audios.m_Sfxs.FirstOrDefault(x => x.name == name);
```

```
    PlaySfx(sfx);
}
public void PlaySFX(int id)
{
    var sfx = audios.m_Sfxs.FirstOrDefault(x => x.id == id);
    PlaySfx(sfx);
}
public void PlaySFX(int id,float pitch)
{
    var sfx = audios.m_Sfxs.FirstOrDefault(x => x.id == id);
    PlaySfx(sfx,pitch);
}
void SetAudioState(bool state)
{
    audioVolume = state ? 1 : 0;
}
void PlaySfx(AudioModel audioModel, float pitch)
{
    if (!isAudioEnabled || audioModel == null)
        return;
    sfxStandartSource.pitch = audioModel.pitch;
    sfxStandartSource.PlayOneShot(audioModel.clip, audioModel.volume);
}
void PlaySfx(AudioModel audioModel)
{
    if (!isAudioEnabled || audioModel == null)
        return;
    sfxStandartSource.pitch = audioModel.pitch;
    sfxStandartSource.PlayOneShot(audioModel.clip, audioModel.volume);
}
void PlayMusic(AudioModel audioModel, bool fadeOut, float fadeTime)
{
    if (!isAudioEnabled)
        return;
    if (fadeOut)
    {
        StartCoroutine(BlendMusic(currentMusic, audioModel, fadeTime));
    }
}
public void StopMusic()
{
    musicSource.Stop();
}
IEnumerator BlendMusic(AudioModel first, AudioModel second, float time)
{
```

```
            yield return null;
            int loopCount = Mathf.FloorToInt(time / 0.1f);
            float firstVolDecreaseAmt = -(first.volume * 2) / loopCount;
            float secondVolIncrAmt = (second.volume * 2) / loopCount;
            float volDelta = firstVolDecreaseAmt;
            while (loopCount > 0)
            {
               musicSource.volume += volDelta;
               if (Mathf.Approximately(musicSource.volume, 0))
               {
                  musicSource.clip = second.clip;
                  currentMusic = second;
                  musicSource.pitch = second.pitch;
                  musicSource.loop = true;
                  volDelta = secondVolIncrAmt;
               }
               loopCount--;
               yield return new WaitForSeconds(0.1f);
            }
         }
         public void HandleDontDestroy()
         {
            transform.SetParent(null);
            DontDestroyOnLoad(gameObject);
         }
      }
//Sesleri düzenli halde tutmak için kullanılan kod parçası

[CreateAssetMenu(menuName = "Create AudioClipArray", fileName =
"NewAudioClipArray", order = 0)]
      public class Audios : ScriptableObject
      {
         [field: SerializeField] public AudioModel[] m_Sfxs   { get; private set; }
         [field: SerializeField] public AudioModel[] m_Musics { get; private set; }
      }
      [Serializable]
      public class AudioModel
      {
         [field: SerializeField] public int       id     { get; private set; }
         [field: SerializeField] public float     volume { get; private set; }
         [field: SerializeField] public float     pitch  { get; private set; }
         [field: SerializeField] public string    name   { get; private set; }
         [field: SerializeField] public AudioClip clip   { get; private set; }
      }
```

```
// Oyundaki verileri kontrol eden parça
public static class DataHandler
{
    private const string levelIndexKey = "LevelIndex";
    private const string totalScoreKey = "TotalScore";
    private const string audioKey = " audio";
    public static int LevelIndex
    {
        get
        {
            return PlayerPrefs.GetInt(levelIndexKey, 0);
        }
        set
        {
            PlayerPrefs.SetInt(levelIndexKey, value);
        }
    }
    public static int TotalScore
    {
        get
        {
            return PlayerPrefs.GetInt(totalScoreKey, 0);
        }
        set
        {
            PlayerPrefs.SetInt(totalScoreKey, value);
        }
    }

    public static bool IsAudioOn
    {
        get
        {
            return PlayerPrefs.GetInt(audioKey, 1) != 0;
        }
        set
        {
            PlayerPrefs.SetInt(audioKey, value ? 1 : 0);
        }
    }
}


// Transform işlemleri için yardımcı
```

```
public class TweenHandler
{
   public void Move(Transform moveItem, Transform movePosition, float time)
   {
      moveItem.DOMove(movePosition.position, time).SetEase(Ease.InSine);
   }
   public void Scale(Transform item, float target, float time)
   {
      item.DOScale(Vector3.one * target, time).SetEase(Ease.OutSine);
   }
}


public abstract class BaseService<T> : MonoBehaviour, IService where T : IService
   {
      private bool _isInitialized = false;
      protected virtual void OnDisable()
      {
         ServiceProvider.RemoveService<T>(this);
         DeInitialize();
      }
      public void Init()
      {
         if (_isInitialized) return;
         _isInitialized = true;

         ServiceProvider.AddService<T>(this);
         Initialize();
      }
      public virtual void Initialize() { }
      public virtual void DeInitialize() { }
   }

public interface IService
   {
      void Initialize();
      void DeInitialize();
   }

public static class ServiceProvider
   {
      private static readonly Dictionary<Type, object> Services = new Dictionary<Type,
object>();
      public static void AddService<T>(object service) where  T : IService
      {
         if (Services.ContainsKey(typeof(T)))
```

```
        {
            Debug.LogWarning($"{typeof(T)} service is already registered!");
            return;
        }
        Services.Add(typeof(T), service);
    }
    public static T GetService<T>() where  T : IService
    {
        bool isExists = Services.TryGetValue(typeof(T), out object value);
        if (isExists) return (T)value;
        BaseService<T> service = Object.FindObjectOfType<BaseService<T>>();
        if (service == null)
        {
            Debug.LogError($"There is no service like : {typeof(T)}!");
            return default;
        }
        service.Init();
        Services.TryGetValue(typeof(T), out value);
        return (T)value;
    }
    public static bool HasService<T>() where T : IService
    {
        return Services.TryGetValue(typeof(T), out object value);
    }
            public static void RemoveService<T>(object service) where T : IService
    {
        if (Services.ContainsKey(typeof(T))) Services.Remove(typeof(T));
    }
}
```