Bir Platform Oyununa Kullanıcı Performansı Temelinde Yapay Zeka Uyarlaması

Sercan Türkmen, Hilmi Yalın Mungan ve Selma Tekir

İzmir Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği Bölümü 35430 Urla/İzmir

Özet Oyun programlama, video oyunlarının yazılım geliştirme bölümüdür. Diğer yazılımlardan farklı olarak oyun içindeki nesnelerin sürekli güncellenmesini gerektirmektedir. Güncelleme işlevinde, nesnenin dünya içinde bulunduğu yer, hız, ivme gibi fiziksel özellikleri, çarpışma işlemleri, animasyon güncellemeleri ve kullanıcı girdisinin ele alınması gibi çok çeşitli işlemler kapsanmaktadır. Yüksek güncelleme frekansı gereksinimi de dikkate alındığında yazılan kodun performansı ve kalitesi ön plana çıkmaktadır. Oyun alanı, yazılım karakteristiklerinden kullanılabilirliğin ötesinde kullanıcının eğlenmesini sağlamayı hedeflemektedir. Yapay zekanın uygulama alanlarının ve tekniklerinin gelişmesi oyunların eğlendirici yönünü arttırmaktadır. Bu çalışmada, bir platform oyunu (Dawn) geliştirilerek oyun içerisindeki kurguyu, geçerli kullanıcıya göre uyarlayan bir yapay zeka entegre edilmesi amacıyla platform oyununu karakterize edebilecek öznitelikler çıkarılmış ve ölçülmüştür. Genel olarak, çıkarılan öznitelikler girdi ve çıktı öznitelikleri olarak gruplandırılarak girdi özniteliklerinin çıktı öznitelikleri ile ilişkisi ortaya konmaya çalışılmıştır. Belirlenen en temel çıktı özniteliği, kullanıcı performansıdır. Kullanıcı performansının ölçümünde bölüm tamamlanma zamanı, kahramanın ölüm nedeni ve bölümlerde uğradığı zarar öznitelikleri baz alınmıştır. Sistem, bu sayede bölüm içerisindeki düşman seçimini ve bir sonraki bölüm önerisini kullanıcının performansına göre belirlemektedir.

Anahtar Kelimeler: Yapay zeka, Rakip Modeli, Öznitelik, Kullanıcı Performansı, Zorluk Derecesi

1 Giriş

Oyun programlama, video oyunlarının yazılım geliştirme bölümüdür. Yazılım mühendisliği üzerinde önemli çalışmalar gerektiren oyun programlama; grafik tasarımı ile birlikte varlık sistemi, kullanıcı arayüzü, fizik motoru, girdi işleyicisi, yapay zeka bileşeni, oyun mantığı, seviye ve ses sistemlerinin geliştirilerek bir bütün oluşturacak şekilde biraraya getirilmesidir [1].

Oyun programlama, diğer yazılımlardan farklı olarak oyun içindeki nesnelerin sürekli güncellenmesini gerektirmektedir. Bu işlem için ideal süre saniyede 60 kere olarak tanımlanmıştır. Güncelleme işlemi genel olarak her model nesnesine ait "update" fonksiyonu ile yapılmaktadır. Bu fonksiyonda nesnenin dünya içinde bulunduğu yer, hız, ivme gibi fiziksel özellikleri, çarpışma işlemleri, animasyon güncellemeleri ve kullanıcı girdisinin ele alınması gibi çok çeşitli işlemler

kapsanmaktadır. Dolayısıyla bu karmaşık işlevin gerçekleştiriminde güncelleme frekansı da dikkate alındığında yazılan kodun performansı ve kalitesi ön plana çıkmaktadır. Hedeflenen güncelleme süresine ulaşılamaması, kullanıcının oyunda donmalar ile karşılaşmasına neden olacaktır.

Oyun alanı, yazılım karakteristiklerinden kullanılabilirliğin ötesinde, kullanıcının eğlenmesini sağlamayı hedeflemektedir. Çok oyunculu ilk video oyunlarından tek oyunculu popüler video oyunlarına geçiş süreci yapay zekanın uygulama alanlarının ve tekniklerinin gelişmesiyle açıklanabilir. Zira yapay zekanın varlığı, oyunun eğlendiriciliğini korumakta hatta artırmaktadır. 1990'lı yılların sonlarına doğru oyunlara entegre edilen ilk yapay zeka uygulamaları kullanıcının oyun içerisindeki konumuna göre, daha önce belirlenmiş bir çözüm kümesinden oyuncuya karşı kullanılacak bir eylemin seçilip uygulanmasını içermekteydi. Daha sonraları yaratılan yapay zekalarda ise yapay zekaya kullanıcının belirli bir özelliği aktarılarak etkisi arttırılmaya çalışılmıştır.

Bu çalışmada, bir platform oyunu (Dawn) geliştirilerek oyun, kullanıcıların oynayış performanslarına göre eylem belirleyen bir yapay zeka ile donatılmaya çalışılmıştır. Sözkonusu yapay zeka bileşeni için ilk olarak eylem bileşeni (çıktı özniteliği) ve onunla nedensellik ilişkisi içerisinde bulunabilecek girdi öznitelikleri belirlenip ölçüm yapılmıştır. Ölçme sonucunda, hesaplanan girdi ve çıktı öznitelikleri arasındaki formulasyon (öğrenme modeli) çıkartılmıştır. Bu formulasyon yeni oynama performanslarında oyuncuya karşı eylemin bir başka ifade ile gerekli çıktının o andaki girdi değerlerine göre hesaplanmasında kullanılacaktır. Bu şekilde yapay zeka entegre edilmiş oyun, müşteri memnuniyetini ve eğlendirici yönünü arttırmaktadır.

Bildirinin devamında, ilk olarak literatürdeki benzer çalışmalardan bahsedilmiştir. Daha sonra ise gerçekleştirilen oyunda yapay zekanın uygulanma şekli ve uygulanırken kullanılan girdi özniteliklerinin belirlenmesinde kullanılan kriterler anlatılmıştır. Bir sonraki bölümde, geliştirilen yapay zekanın ayrıntıları, uygulamada karşılaşılabilecek sorunlar ve bu sorunların nasıl aşılabileceği aktarılmıştır. Sonuç bölümünde ise elde edilen sonuçlar ve kazanımlar tartışılmıştır.

2 Literatür

Oyun yapay zekası, en genel tanımıyla oyunlara bir çeşit yapay zekanın entegre edilmesidir. Oyun yapay zekası terimi, oyun içerisinde çarpışma tespitinin yapılması gibi nispeten sınırlı bir işlevi karşılamak için kullanılabilirken oyunda bir sinir ağı algoritmasının uygulanması gibi daha karmaşık bir gerçekleştirimin varlığını da kastediyor olabilir. Bu çalışmada, belirtilen iki uç tanım arasında yer alan bir yapay zeka tanımı benimsenmektedir [2].

Video oyunlarının eğlendiriciliğinin artırılması için oyunlara yapay zeka entegre edilmektedir. Oyuna entegre edilecek yapay zeka için oyuncu davranışının modellenmesi vazgeçilmez bir gereksinimdir çünkü temel hedef oyuncuyu yenmekten ziyade onun eğlenmesini sağlamaktır. Oyunlarda oyuncu davranışı modellenirken farklı bilişsel seviyeler dikkate alınmaktadır. Oyuncu profilinin çıkartılması en üst bilişsel seviyeye karşılık gelirken diğer seviyeler sırasıyla oyuncu-

nun strateji, taktik ve eylem davranış modellerinin oluşturulmasını içermektedir. Oyuncu profilinden eylem davranış modeline doğru ilerlerken modelin kapsayıcılığı, hassasiyeti, genel geçerliği azalırken doğrudan ölçülebilir eylemler nedeniyle oluşturulması daha kolay hale gelmektedir [3].

Video oyunlarında oyuncu modelini kullanan yapay zeka üç farklı rolden birini üstlenmektedir: Antrenör (koç) rolü, rakip rolü ve arkadaş rolü. Antrenör rolü, bu modeller arasında en az ayrıntılı olanıdır. Bu modeldeki bir yapay zekanın yapması gereken oyuncuya, oyunun temel mekaniklerini göstermek ve oyuncuyu yönlendirmektir. Dolayısıyla geliştirilecek bu türde bir yapay zekanın statik ya da dinamik olması kullanıcı açısından, oyun deneyimi için fark yaratmayacaktır. Rakip rolündeki oyun yapay zekası ise, oyunun güçlük seviyesini oyuncunun yeteneğine ve oyuncunun oynama stiline uygun şekilde düzenlemeye çalışır. Yapılan araştırmalar rakip rolünün doğru bir şekilde derecelendirilmesinin önemini vurgulamaktadır. Zira rakip karakterlerin çok zayıf olması oyuna ilginin kaybedilmesine neden olurken çok güçlü rakipler oyuncuyu yıldırarak oyundan çıkarır. Arkadaş rolü de rakip rolüne benzerlik göstermektedir. Arkadaş rolündeki yapay zeka, kullanıcı ile birlikte çalışarak, rakip rolündeki yapay zekaya karşı eylem almaktadır. Bu sebeple geliştirilen yapay zeka, kullanıcının aldığı eylemleri yorumlayarak ve alacağı eylemleri tahmin ederek hareket etmelidir [3].

Oyun yapay zekasına rakip modeli dahil edilirken oyun ortamında gözlemlenecek öznitelikler belirlenir. Ölçülen öznitelikler kullanılarak o andaki oynama performansını değerlendirecek bir değerlendirme fonksiyonuna gereksinim vardır. Bu değerlendirme fonksiyonu çıktısı ve mevcut rakip modeli oyuna yapılacak uyarlama için kullanılacak girdilerdir. Bu şekilde bir örnek olay tabanlı oyun yapay zeka uyarlaması Bakkes et al.'da [4] görülmektedir.

Video oyunlarında oyunun güçlük seviyesinin uygun şekilde ayarlanması iyi bir oyun tasarımının vazgeçilmezlerindendir. Oyunun güçlük seviyesi belirlenirken oyun dinamik yapısının bir kenara bırakılıp oyuncunun bakış açısına odaklanılması doğru bir yaklaşımdır. Bir oyundaki güçlük seviyesi önceden tanımlanmış sorunlar (challenge) kombinasyonu ya da dizisi üzerinden tanımlanır. Oyuncunun her bir sorunun üstesinden gelirken yaşadığı kesikli güçlük değerleri, oyuncunun öğrenme eğrisinin tahminlenmesini sağlar. Amaç, bu kesikli güçlük değerlerinin oyuncunun yeteneğini, öğrenme eğrisini iyi bir şekilde temsil ediyor olmasıdır [5].

3 Yapay Zekanın Uygulanması

Video oyunlarına, farklı eylem türlerine sahip olan yapay zekalar eklenebilir. Bu yapay zekalar, kullanıcıyı yönlendirmek, kullanıcıya yardım etmek ya da kullanıcıya karşı olmak şeklinde farklı eylemlerle donatılmıştır. Her bir farklı yapay zeka türünün, kullanıcının oyundan aldığı memnuniyeti olabilecek en yüksek seviyeye çıkartmak için, ayarlanmaları gereken belirli bir etkileşim seviyeleri vardır. Dawn için uyguladığımız yapay zeka, sadece kullanıcıya karşı eylem almaktadır.

Bu çalışmada gerçekleştirilen yapay zeka uyarlaması, oyuncu davranış modelini çıkarırken oyuncunun doğrudan ölçülebilir eylemlerini baz aldığından oyuncu bilişsel seviyesi eylem düzeyinde temsil edilmektedir.

Bu tür yapay zekanın, kullanıcı memnuniyetini en yüksek seviyeye çıkartabilmesi için, zorluğunun, kullanıcının deneyimi seviyesinde olması gerekmektedir. Bu seviyenin doğru ayarlanması çok önemlidir, zira bu seviye çok yüksekse, kullanıcı yapay zekaya karşı başarısız olmaktan bıkıp oyunu bırakabilir veya bu seviye çok düşükse, kullanıcı oyunu eğlendirici ve zorlayıcı bulmayıp yine aynı şekilde oyunu oynamaktan vazgeçebilir. Uygulanacak yapay zekadaki en uygun seviye, kullanıcının sahip olduğu tecrübenin seviyesidir. Yani, yapay zekanın seviyesi, kullanıcının performansına göre ölçülmelidir.

3.1 Yapay Zekanın Geliştirildiği Ortam ve Kullanılan Altyapılar

Geliştirilen yapay zeka "haxe" dili [6] ve "flixel" [7] kütüphanesi kullanılarak prototip bir oyun üzerinde geliştirilmiştir. "Haxe" yüksek seviyeli, açık kaynak bir programlama dili ve derleyicidir. "Flixel" kütüphanesi ise özellikle hızlı prototip geliştirmek için uygun olup "haxe" dilinin sağladığı çoklu platformlara derlenme özelliği sayesinde oyun farklı platformlara tek kod altyapısından (code base) derlenebilmektedir. Kod düzenleyici olarak "FlashDevelop" kullanılmıştır. Nesneye dayalı yaklaşım izlenip, "Bitbucket" üzerinden versiyon kontrolü yapılmıştır.

3.2 Oyuncunun Ölçülen Öznitelikleri

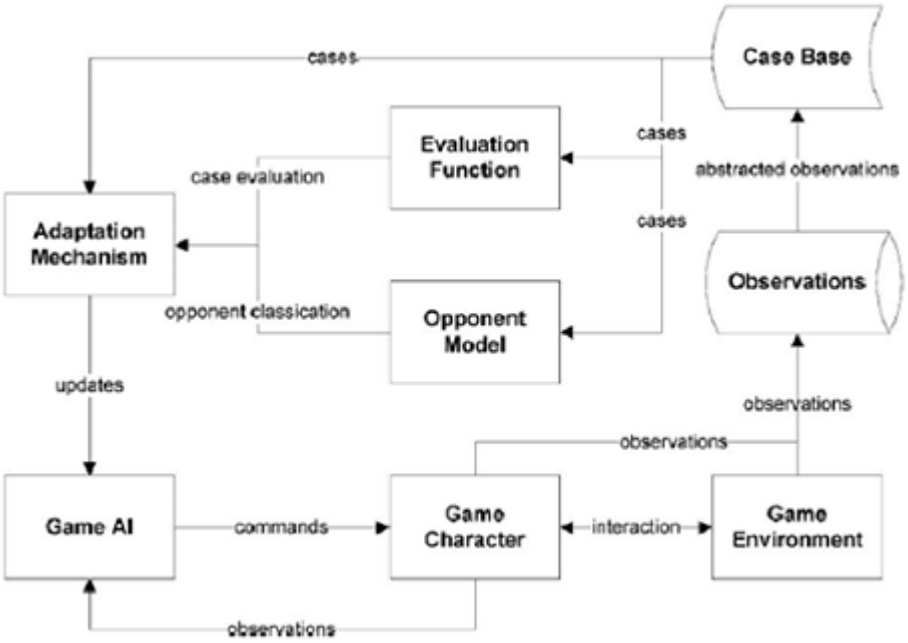
Dawn içindeki oyuncu performansı, belirli girdi öznitelikleri birleştirilerek elde edilmiş bir veridir. Ancak bu girdi özniteliklerinin saptanmasında belirli kriterler olması gerekir. Bu kriterler belirlendiğinde, sonuçta ortaya çıkan girdi özniteliklerinin sayısı ne kadar fazla olursa, yapay zekanın hassasiyeti ve bunun sonucunda belirlenen oyuncu performansı o kadar doğru hesaplanabilir. Ancak bu sayının fazla olması oyun süresince kullanıcı performansı hesaplanırken oluşacak işlem yükü dolayısı ile, altbölüm geçişleri sırasında ya da belirli özel altbölümlerdeki veri değişimi sırasında oluşabilecek işlem yavaşlığı, programın ulaşabileceği kullanıcı sayısından ödün verebilir. Bu tür bir ödünün verilmesi, kullanıcının oyundan sıkılıp bırakmasına yol açabilir. Bu sebeple veri akışının oyunu yavaşlatmaması için girdi özniteliklerinin belirli bir dengeyle saptanması gerekmektedir. Oyun için kullanıcı performansını en iyi şekilde belirleyen öznitelikler seçilmelidir. Dawn için bu girdi öznitelikleri;

- oyuncunun altbölüm içinde düşmanlardan aldığı zarar,
- altbölüm tamamlama süresi,
- altbölüm içindeki tüm düşmanları yok etme süresi ve
- hangi tür düşmandan daha çok zarar gördüğüdür.

Bu özniteliklerden kullanıcının altbölümü tamamlama süresi, oyunun rekabetçi yanını arttırmak için seçilmiştir. Kullanıcının bölümü bitirme süresi, ortalamanın altında bir süre ise kullanıcı performansı artacak, üzerinde ise azalacaktır. Bu artma ve azalma değeri oyuncu performansına oransal bir etki ile hesaplanmaktadır.

3.3 Metodoloji

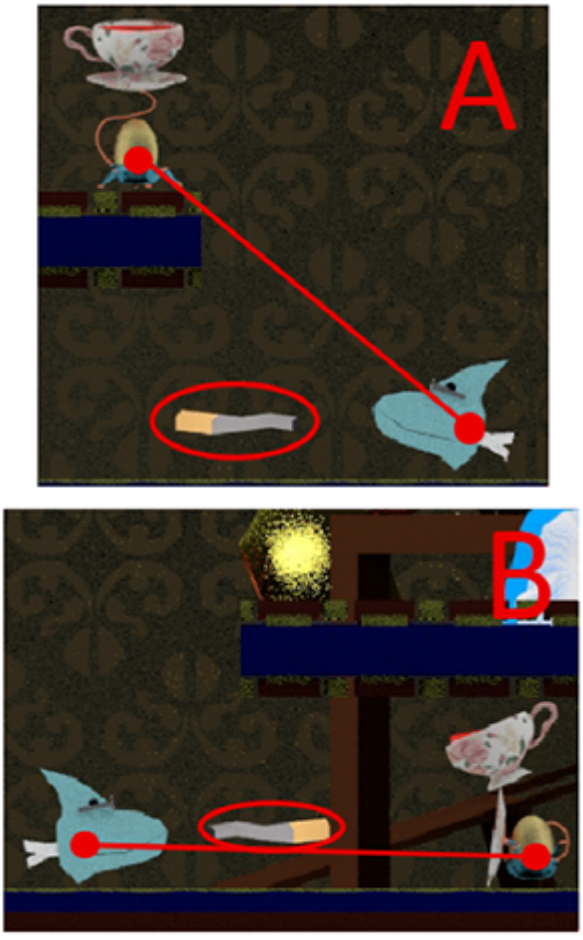
Bu çalışmada, Bakkes at al. [4] tarafından önerilen örnek olay tabanlı oyun yapay zeka modeli belirlenen özniteliklere uyarlanmıştır. Dawn'da, Şekil 1'deki rakip modeli, oyun içerisindeki düşmanların kullanıcıya karşı aldıkları eylemlerin özelliklerini yansıtmaktadır. Değerlendirme fonksiyonu, kullanıcının aldığı eylemlerdeki başarı oranını ölçerek, rakip modeli kısmı ile birlikte, uyarlama mekanizmasına etki eder. Uyarlama mekanizmasında ise, kullanıcının performansı temelinde rakip modelindeki özellikler tekrar ayarlanarak yapay zeka güncellenmiş olur.



Şekil 1. Örnek olay tabanlı oyun yapay zeka uyarlaması modeli [4].

Şekil 1'de görülen yapay zeka uyarlaması modelinin, rakip bir yapay zeka olan "turret"e uygulanması, Şekil 2'de gösterilmiştir. Prototip oyununda geliştirilen "Turret" düşmanı belirli aralıklar ile baktığı yöne doğru ateş etmektedir. Şekil 2'nin ilk kısmında görüldüğü şekilde "turret", oyuncunun bulunduğu yöne belirli aralıklarla ateş etmektedir. "Turret"in bu özelliği rakip modelini belirlemektedir. Değerlendirme fonksiyonu, kod içerisinde her saniyede 60 defa çağrılan "update" fonksiyonunun içine yerleştirilerek, oyuncunun rakip modeli tanımlayan özellikleri değiştikçe, uygulama mekanizmasını etkiler. Bu durumda Şekil 2 ikinci kısma geçilirken, oyuncu ve "turret" arasındaki açı hesaplanarak, "turret" oyun-

cunun bulunduğu tarafa döner ve oyuncu performansı değişirse, "turret"in son ateşinden itibaren geçen süreyi hesaplayarak, ateş sıklığını arttırır ya da azaltır.



Şekil 2. Turret düşmanına yapay zeka uyarlaması.

Oyuncu performansına etki eden ilk faktör altbölüm bitirme süresinin ölçülerek değerlendirilmesi şeklindedir. Geliştirilen oyun bir platform oyunu olduğu için altbölüm bitirme süresi oyuncu performansı hakkında yüksek öncelikli olarak bilgi sağlayacaktır. Altbölümleri bitirmek için gereken ortalama süre çeşitli şekillerde belirlenebilir. Prototip olarak geliştirilen oyunda bu süreler yaklaşık değerler olarak bölümdeki düşman sayısı ve engellere göre belirlenmiştir. Fakat bu sürenin, her bölüm için yeteri kadar kullanıcı ile test edilip belirlenmesi daha uygun olacaktır. Ayrıca altbölümlerde alınan zarar belli bir sınırın altında ise kul-

```
//update player performance
var delta:Float = completionTime - stageStatistics.startTime;
var ssInfo:Dynamic = subStages[currentSubStageIndex].substageInfo;
if (delta - ssInfo.averageCompletionTime > 10)
    Reg.playerPerformance -= Reg.playerPerformance / R.BAD_SUBSTAGE_TIME;
else if(delta - ssInfo.averageCompletionTime < 0)
    Reg.playerPerformance += Reg.playerPerformance / R.GOOD_SUBSTAGE_TIME;
```

Şekil 3. Oyuncu performansının güncellenmesi.

```
substageDyn = substagePool[0];
var closestDifficulty:Float = Math.abs(substageDyn.difficulty - Reg.playerPerformance);
for (s in substagePool) {
    if (Math.abs(s.difficulty - Reg.playerPerformance) < closestDifficulty) {
        // pick closest performance level to user
        substageDyn = s;
        closestDifficulty = Math.abs(substageDyn.difficulty - Reg.playerPerformance);
    }
}
```

Şekil 4. Altbölüm havuzundan oyuncu performansına en yakın altbölüm seçimi.

lanıcı performansı arttırılacak, bu sınırın üzerinde ise azaltılacaktır. Altbölümlerdeki tüm düşmanları öldürme süresi ise her bir altbölümü geçmek için o altbölümdeki tüm düşmanları öldürene kadar geçen süredir. Bu bilgiyle de aynı şekilde oyuncu performansı üzerinde oransal bir oynama yapılmıştır (Şekil 3).

Dawn içindeki yapay zeka, oyuncu performansını belirtilen verilere göre ölçüp, kullanıcının tecrübe ve yetenek verilerine dayanarak, kullanıcıyı zorlayacak ama kullanıcının başarılı olabileceği altbölümleri belirlemektedir. Altbölüm seçimi yapılırken öncelikle bölüm bilgisini içeren dosya okunur. Bu dosyada bölümü bitirmek için geçilmesi gereken altbölüm sayısı, bölüm sonu canavarının altbölüm bilgisi ve altbölüm havuzu bulunmaktadır. Altbölüm havuzu dizisinde altbölümün ismi, zorluk derecesi, ortalama bitirme süresi gibi bilgiler yer almaktadır. Oyuncu bir altbölümü tamamladığında diger altbölüm seçimi yapılırken altbölüm havuzundaki zorluk dereceleri arasında oyuncu performansına en yakın komşu değere sahip altbölüm bulunarak oyuncunun kendi seviyesinde bir bölüm ile karşılaşması hedeflenmektedir (Şekil 4).

3.4 Performans Girdilerinin Yapay Zekaya Yansıtılması

Performans girdisi yapay zekaya yansıtılırken, performans girdisi düşmanın belirlenmiş özniteliklerini arttırmak ya da azaltmak için bir oran olarak kullanılmıştır. Buna örnek olarak prototip oyununda geliştirilen “Turret” düşmanı verilebilir. Bu düşman belirli aralıklar ile baktığı yöne doğru ateş etmektedir. Burada düşmanın ateş etme sıklığı, normal değeri ile oyuncu performansının ters orantısı alınarak güncellenmiştir. Bu sayede performansı yüksek olan bir oyuncu karşısına, daha sık ateş eden, yani oyuncunun yeteneklerine daha uygun bir düşman çıkartılması hedeflenmiştir (Şekil 5).

Bu girdinin oran olarak kullanıldığı başka niteliklerden örnek vermek gerekirse bunlar, düşmanın hareket hızı (Şekil 6), uyguladığı zarar miktarı gibi özel-

```
lastShotTime += elapsed;
if ( lastShotTime > shootTimer) {
    var bullet:Bullet = cast(bullets.recycle(Bullet), Bullet);
    bullet.setGraphic(AssetPaths.sigara__png);
    bullet.shoot(midpoint, facing, Turret);
    lastShotTime = 0;
    shootTimer = R.FREQ_TURRET_SHOOT / Reg.playerPerformance / 2;
    animation.play("shoot");
}
```

Şekil 5. Turret'in ateş etme sıklığının oyuncu performansına göre güncellenmesi.

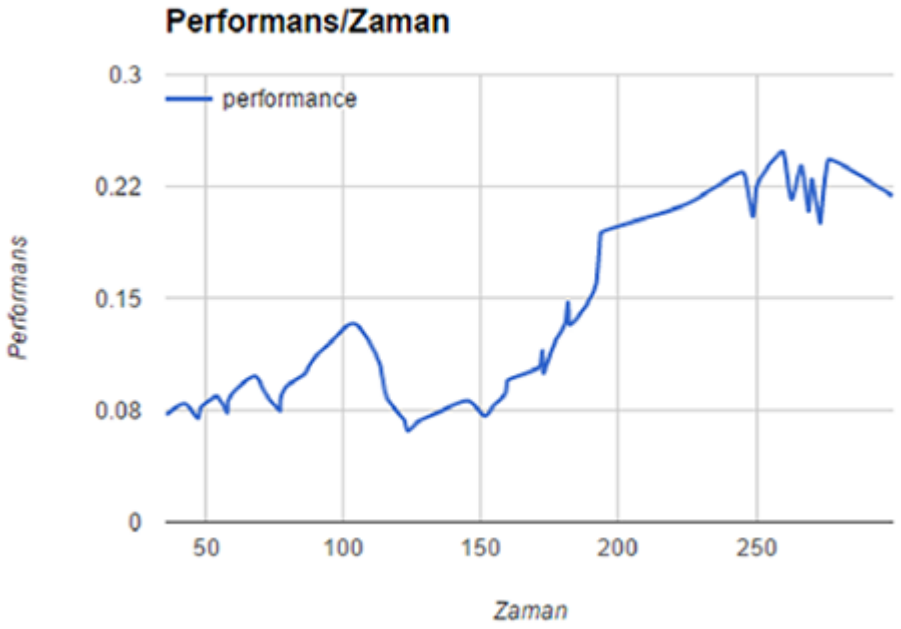
```
if (angleBetweenPlayer() > 0 ) {
    pivot.acceleration.x = 500 * Reg.playerPerformance;
    flipX = false;
}else {
    pivot.acceleration.x = -500 * Reg.playerPerformance;
    flipX = true;
}
```

Şekil 6. Düşmanın hareket hızının oyuncu performansına göre güncellenmesi.

```
// turns off bullet attack if player performance lower then 0.3
if(Reg.playerPerformance > 0.3){
    var angleBetween:Float = angleBetweenPlayer();
    bulletTimer -= elapsed;
    if (bulletTimer < 0) {
        bulletTimer = R.FREQ_TURRET_SHOOT / Reg.playerPerformance / 2;
        var bullet:Bullet = cast(bullets.recycle(Bullet), Bullet);
        bullet.setGraphic(AssetPaths.sigara__png);
        bullet.shootWithAngle(midpoint, angleBetween, FredericBody);
        FlxG.sound.play(AssetPaths.boss_attack__mp3);
    }
}
```

Şekil 7. Bölüm sonu canavarının ateş etme özelliğinin açılıp kapatılması.

liklerdir. Ayrıca bu performans girdisi sadece oran olarak değil aynı zamanda düşmanın bazı özelliklerini kapatmak veya açmak için de kullanılmıştır. Şekil 7'de görüldüğü üzere prototipte geliştirilen bölüm sonu canavarının ateş etme özelliği oyuncu performansı belirli bir değerin altında olduğunda kapatılmıştır. Ancak not edilmelidir ki bölüm sonu canavarı ile o an savaşan oyuncunun performansını isabetli atışlar yaparak arttırması halinde kapatılan özellik performans arttığı anda dinamik olarak açılacak ve ateş etme sıklığı eniyileştirilecektir. Böylelikle oyun seyrederken eş zamanlı olarak zorluk derecesi ayarlanmaktadır.



Şekil 8. Oyun süresince bir kullanıcıya ait performans-zaman grafiği.

4 Sonuç

Sonuç olarak, izlenen yöntem doğrultusunda geliştirilen prototip oyun, zorluk derecesini oyun başlamadan oyuncuya sormak yerine dinamik olarak kendisi belirlemektedir. Oyuncu oyun içinde düşmanlara ve oyun mekanisklerine alıştıkça değişen oyuncu seviyesine göre oyun kendini zorlaştırmakta, oyuncuya zor geldiği zamanlarda ise kendisini dinamik olarak kolaylaştırmaktadır. Uyarlanan yapay zekanın çok sayıda oynama performansı verisini işleyerek oyunun zorluk seviyesini daha iyi dengelemesi mümkündür.

Ayrıca farklı türde bir oyun için oyuna uygun öznitelikler seçilerek benzer bir metodoloji uygulanarak yapay zeka uyarlaması gerçekleştirilebilir.

Şekil 8'de, elde edilmiş olan grafik, bir kullanıcının oyun süresince elde edilen performans zaman grafiğidir. Grafikte, oyuncu performansının başlangıç değeri 0.1'dir. Kullanıcı, 100. saniye civarlarına kadar, performansının düşük olmasından, bazı kolay altbölümler ile karşılaşmış ve performansını arttırmıştır. Ancak 100. ve 150. saniye aralığında, artan performans değerinden dolayı, bazı altbölümler zor gelmiş ve kullanıcı performansı düşmüştür. 150 ve 250 aralığında, kullanıcının oyuna alışarak, performansını arttırdığı görülmektedir. 250. saniyeden sonra ise, kullanıcının performans değeri, yaklaşık sabit bir değer alarak ilerlemektedir ve bu da kullanıcının ulaştığı istikrarlı performans değerini göstermektedir. Eğer oyuncu, oturumu kapatmadan tekrar oyuna başlamak isterse, başlangıç değeri olarak 0.22 alınacaktır.

Kaynaklar

1. Oehlke, Andreas, "Learning Libgdx Game Development", Packt Publishing, 2013.
2. Bourg, David M. and Seemann, Glenn, "AI for Game Developers," O'Reilly Media, Inc., 2004.
3. S. C. J. Bakkes, P. H. M. Spronck, and G. van Lankveld, "Player behavioural modelling for video games," Entertainment Computing, vol. 3, pp. 71-79, 2012.
4. S. C. J. Bakkes, P. H. M. Spronck, and H. Jaap van den Herik, "Opponent modelling for case-based adaptive game AI," Entertainment Computing, vol. 1, pp. 27-37, 2009.
5. M.-V. Aponte, G. Levieux, and S. Natkin, "Measuring the level of difficulty in single player video games," Entertainment Computing, vol. 2, pp. 205-213, 2011.
6. Haxe Foundation, [http://haxe.org/manual/introduction-what-is-haxe.html]: para. 1[Haziran 29, 2015]
7. HaxeFlixel, [https://github.com/HaxeFlixel/flixel] [Haziran 29, 2015]