172114203 & 152113001

Unity3D Lightweight Render Pipeline ile MiniMax Algoritması kullanan Yapay Zeka ile Mirror Networking tabanlı çok oyunculu mobil ve bilgisayar destekli strateji oyunu

---

Düzce Üniversitesi 2020 Yaz Okulu Tez Projesi

Özgür Özbek & Atilla Çoruhlu

20.08.2020

Tez projesi için sunulan en son geliştirmeleri ve teknolojileri kullanarak çok oyunculu bir oyun yapımı hakkında gerekenlerin temel olarak açıklanıp çeşitli sorunların nasıl çözüldüğünün açıklanması.

Kullanılan Teknolojiler, Araçlar ve Açıklamaları

Unity3D: Bir oyun motoru. Oyun yapmak için gereken ses sürücüleri, ışık sekmesi takibi yazılımı, kod derleyici, ekrana çizdirme, sürücü takibi gibi bir çok yazılımı içerisinde barındıran bir motor yazılım. Bu dökümanda Unity olarak geçecek. Unity araçları ve metodları Türkçe'leştirilmeyecek ve asıl isimlerini koruyacak.

Visual Studio Code & Monospace: Kod yazmak için kullanılan çalışma ortamınına bağlı çalışan, içerilerinde olan Linter, Parser, Error Handling ve benzeri kod yazmayı kolaylaştıran yazılımlar ile bir entegre geliştirme ortamı.

Adobe Illustrator CS6: Oyun içinde kullanılacak iki boyutlu neredeyse her doku için (buton ikonları, basit yazılar vb.) bir dijital çizim uygulaması.

Lightweight Render Pipeline: Unity içinde sunulan son geliştirmelere sahip, ekran kartı ve işlemciye yük bindirmemek adına önceden yazılım desteği tamamlanmış bir görüntü işleme hattı. Bunun sayesinde Post-Processing, Shaders ve bir çok Rendering efekti kullanabiliyoruz. Bu dökümanda LWRP olarak kısaltılacak.

MiniMax Algoritması: Olabilecek bütün adımları bir iterasyona kadar deneyip en yüksek ve en düşük sonucu elde etmeye yarayan bir yapay zeka algoritması. Oyunda, rakip olmadığı durumlarda rakibin yerine tatmin edici bir sanal rakip oluşturmak için kullanılıyor.

Yazılım: Unity ile haberleşmek için C# ve Java, Dökümantasyon için Markdown, Yapay Zeka ile iletişim kurmak için C# ve Python, Networking için C++, Versiyon takibi için Git kullanılmakta.

Oyun Teorisi: Geliştirme aşamasında bir çok rekabetçi kararı almak için John F. Nash tarafından ileri sürülmüş rasyonel kararları vermeye yardımcı matematiksel formüller kullanıldı.

Steam & Google Play: Oyun geliştirme, sürdürme ve yayınlanması hakkında sunulan bir çok dökümandan faydalanıldı. Geliştirme hem mobil hem bilgisayar için yapılıp yayınlandı.

Asset Store: Çeşitli ses, model ve benzeri ürün elde edebilmek için online olarak editör içerisinde sunulan bir market.

Reaper, Kontakt & Bosca Ceoil: Müzik, orkestrasyon vb ses mühendisliği yaparken ve diğer ses efektlerini oluştururken kullanıldı.

Trello: Proje takibi için bulletin board.

Sahneler ve Açıklamaları

Menu Scene: Oyuna ilk girdiğimizde karşımıza çıkan ekran. Buradan ses açıp kapatmak gibi çeşitli ayarları yapıp, olduğumuz versiyonu görüp, Mobil ise oyuna puan verebilip, oyun lobisi başlatma ekranına geçebildiğimiz veya oyundan çıkabildiğimiz ekran. Oyunun başlığı da bu ekranda yer alıyor.

Lobby Scene: Yeni bir oda kurabiliyor, kurulu odaya katılabiliyor, ve genel olarak Mirror kullanarak multiplayer için sistem şartlarını sağlıyoruz. Versiyonda eğer varsa sanal rakibe karşı offline lobi kurma imkanı sunuluyor.

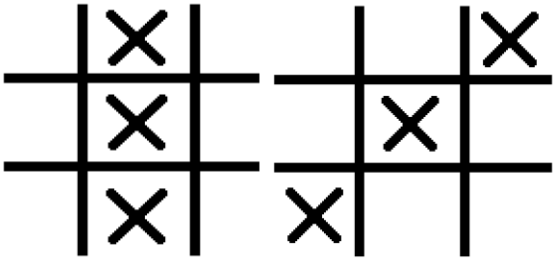
Game Scene: Oyunun temel mantık ekranı.

Credits Scene: Oyun içerisinde kullanılan varsa açık kaynaklı yazılımların kaynaklarının gösterildiği, yazarların isimlerinin verildiği, ve varsa sunulması gereken teşekkürlerin sunulduğu ekran.

Tutorial Scene: Oyunun nasıl oynandığını anlatan bir ekran. Yazılar ve resimler ile desteklenmiş bir Slide Show mantığı ile kullanıcıya basit ama gerekli bilgiler veriliyor.

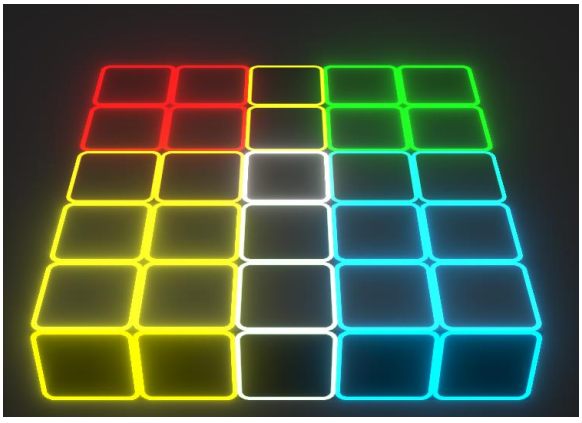
Mantık ve Kurallar

Öncelikle XOX oyunu hakkında bilgi verilmesi gerekiyor. XOX oyunu, 3x3 bir alanda, genellikle kağıt ve kalem ile oynanan bir oyundur. Bir hizada veya çapraz olarak yanyana X veya O getiren oyuncu kazanır. Önce bir oyuncu X yazarak başlar, sonra sırayla O, X, O, X şeklinde devam ederler.



Resimlerde, O'yu yok sayarak X'in kazanma durumunu gösterdik. Bizim oyunumuzda ise, 4 tane XOX masasını üst üste koyup, oyuna biraz daha strateji ve tekrarlanabilirlik kattık. Online multiplayer sağlanabilirse eğer, oyuncu bulmak ile ilgili de sıkıntı olmayacak.

Mantık: Oyun 5x5 bir alanda oynanıyor. Orta kare her masada bir köşe kaplayacak şekilde yerleştirildi.



Bunun dışında normal XOXden farklı olarak, bir oyuncu üçleme yaptığı zaman oyun bitiyordu, fakat bizim durumumuzda bu şekilde değil. En fazla üçlemeyi yapan oyunu kazanıyor.

Açıklar: Masaların birleşmesinden ötürü oluşan 4 temel açık var.

1) İki masada da olan fakat 5x5 tahtada tek görünen yerden alınan puanın sayısı ne olacak?

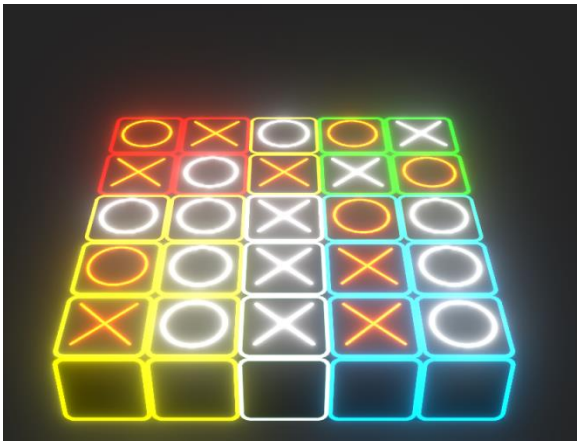
2) Berabere bitme oranını nasıl azaltacağız?

3) İlk başlayanın avantajı nasıl bozulacak?

4) Masayı 3x3 olarak 4 masaya nasıl böleceğiz?

Açıkların Çözümleri: Olabildiğince anlaşılabilir ve basit şekilde bunlar çözülmeli. Oyuncuya anlaması güç çözümler sunup oyunun zevkinden çalmak istemiyoruz.

1) İki masada da olan fakat tek gözüken yerleri belirtmek için, arkaplanda renk ile kategorileştirme yaptık. Böylelikle her kutunun hangi masaya ait olduğu belli oluyor.



Bu durumda da gördüğünüz oyun içi ekran çıktısında bulunan X'lerin 2 tane üçlemesi bulunsa da toplam 3 puanı oluyor. Bu durum maalesef berabere kalma oranını arttırıyor.

2) Berabere bitme oranını azaltmak için, eğer iki oyuncunun da puanları eşitse, puanları topladıkları alanların toplamına bakacağız. Oyunumuzun ekran çıktısında gördüğünüz üzere X'lerin de O'ların da 3 defa üçleme yaptığını doğal olarak 3 puanları olduklarını görüyoruz. Aynı zamanda O'nun puanlarını aldığı toplam

alan 9 iken, X'in puanlarını aldığı toplam alan 5 oluyor. Bunu baz alarak bu durumda O'nun kazandığını kabul ediyoruz.

3) İlk başlayan en fazla 4 masada avantajlı olabildiği için, O oyuncusuna da bu avantajı sağlamamız gerekiyor. O oyuncusu bir turda en fazla 2 masada avantajlı olabileceği için, ilk turunda 2 hamle yapmasına izin veriyoruz.

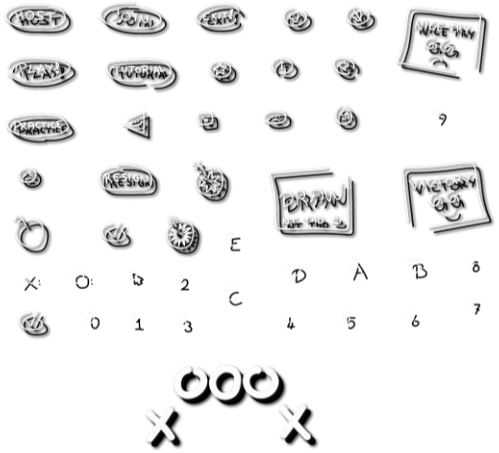
4) 5x5 masayı da, resimde gördüğünüz üzere renk kodu ile ayırarak 4 masaya bölebiliyoruz.

Kurallar: Yukarıda yazan her şey baz alınarak, kurallarımız şu şekilde.

- İki oyuncu ile oynanıyor.
- Önce X hamle yapıyor.
- İkinci turda O, 2 hamle yapıyor.
- 3x3 masaların ortak bölümleri çift puan kazandırıyor.
- Bir oyuncu bir masada puan kazandığında, o masada oyun bitmiyor.
- Bütün masalar dolduğunda en çok puanı olan oyuncu kazanıyor.
- Eğer bütün masalar dolduğunda iki oyuncunun da puanı eşit ise, en çok puan almış kareye sahip oyuncu oyunu kazanıyor.
- Bütün puanlar eşit ise, oyun berabere bitiyor.

Ön Geliştirme Süreci

Önceki resimlerde görülen ekran görüntüleri, gördüğünüz Vector çizimleri vb. tamamen bizim tarafımızdan Unity3D veya Adobe Illustrator içerisinde yapıldı. X ve O için gereken çizim çok basit olsa da objelerin pipeline içerisinde çizilmesini PBRGraph kullanarak Emisyon ayarları ile yaptık. Bütün görüntü ve çizimler oyun planına uygun olacak şekilde tamamlandıktan sonra oyunun ana mantığı programlanmaya başlandı. Bu evrede her masayı tutması için 4 tane çok boyutlu liste kullandık. Böylece ortak bir kare tıklandığında iki listede de doğru yere oturtup ayrı olarak puan takibi yapabilmemiz sağlandı. Bu oldukça kritik bir yaklaşım çünkü eğer tek boyutlu bir liste ile bütün 5x5 masayı hesaplamak isteseydik, bu hem daha yavaş olacaktı hem de 2 katı puan vermek için ekstra kaynaklar kullanmamız gerekecekti.

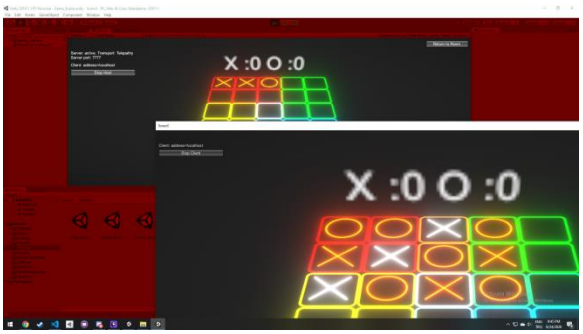


Gördüğünüz resimde fazladan gölgelendirilme kullanıldı, çünkü Unity içerisinde kullandığımız bütün çizim elemanları beyaz renkte olmak zorunda. Emisyon kullandığımız için renk hesabını HDR, Işık sekmesi hesabını da Shaderlara bıraktık. Böylelikle seçtiğimiz herhangi bir hue-saturation değeri veya RGB değeri beyazın üzerinde tamamen doğru olarak gözükecek.

Karelerin üzerine de nesne tabanlı olacak çekilde, ayrı bir katmanda eklediğimiz "Button.cs" scripti bahsettiğimiz listelere veriyi ekleyip "GameMaster.cs" scriptinde de skor hesaplaması yapılıp "Score.cs" scriptinin bağlı olduğu objede gösteriliyor. Sıra, tur dengesi, kuralların koda dökülmesi ve butonlara tekrar tıklanamama gibi mantık dışı hataları giderdikten sonra oyun tek ekranda, 1 fare ile 2 kişilik şekilde oynanabilir hale getirildi.

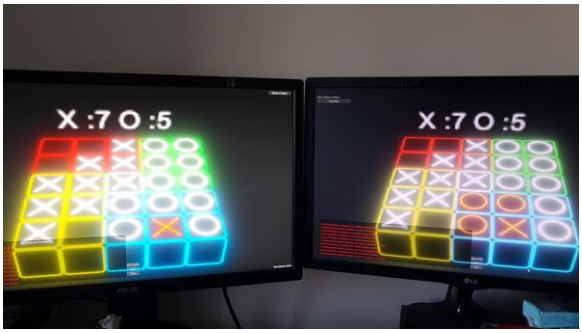
Çok Oyunculu Geliştirme Süreci

Önceki sürümde oluşturulan önemli birkaç değişken var. Bu değişkenlerin çevrimiçi ortamda da senkron halde takip edilmesi gerekiyor. Hızlı ve sürekli olabilmesi için Mirror sadece ilkel veri tiplerini destekliyor. Bunlar string, char, integer, float vb. verilere deniyor. Bir integer olan MoveNumber değişkeni bir ek efor kullanmadan Mirror içerisindeki SyncVar dekoratörü ile senkronize edilebiliyor. Bunu senkronize etsek dahi serverdaki değişiklikleri ve clientteki değişiklikleri birbirinde gösteremiyoruz.



Bunun için önce Mirror içerisindeki senkronize class objesini kendi oluşturduğumuz bir objeye inherit olacak şekilde ekledik. Sonrasında ise kendi objemizi ilkel veri tiplerini tutan bir struct yaptık. Önceden oluşturduğumuz listeleri bu structın içine atadık ve sonraki liste işlemlerini de bu structtan baz alacak şekilde düzenledik. Bu değişiklik bizi yaklaşık 70 saat geriye attı, fakat bunun sayesinde artık hala görünemese dahi client ve server içerisinde oluşan verileri senkron hale getirdik. Materyal değişkeni bir ilkel veri tipi olmadığı için Mirrorda bunu senkron hale getirmek adına yine benzer dönüştürmeleri kullandık.

Client → Server veri aktarımı Authenticator olmadan mümkün değil, ve Mirror bize önceden bahsettiğim gibi sadece ilkel authenticatorler sağlıyor. Bu yüzden öncelikle Clientta çalışacak bütün fonksiyonları Server'a aktardık. Serverda çalışması gereken fonksiyonları ilkel bir authenticatorün arkasına depoladık. Client'e asla bir authenticator vermediğimiz için Server tarafının sadece Serverda çalışmasını sağladı. Ek olarak bir katman daha güvenlik önlemi sağlıyor. RPC kullanarak Client'in servera fonksiyon çalıştırmasını öneren metodları yazdık. Bu şekilde şimdi Client ve Server birbiri üzerinde yapılan işlemleri görebiliyor oldu. Server skor güncellemesi, materyal dönüşümü gibi şeyleri de kendisi üzerinde yaptığı için senkronizasyonda hatamız olmadı ve bütün clientlar aynı efektleri ve skoru görebilir oldu. Önceden, RPC kullanmadan ilkel metodlarla yapmak istediğimizde de bir Bilgisayar Mühendisliği problemi olan Yarış Kondisyonu ile karşılaştık. Bu kondisyon özetle iki atomik işlemin, senkron olmak için aynı anda çalışıp bir fonksiyonun karmaşıklığının diğerinden bir kare fazla olması ve sonuç olarak bir mantık işlemi yapılmak istediğinde de yavaş çalışıp senkronun bozulmasına denir. Bu hatanın farkedilip düzeltilmesi gerçekten zahmetli olsa da RPC ve Unity zamanlayıcıları bize olağanüstü bir avantaj sağladı. Gördüğünüz resim, sadece Server efektlerinin Client üzerinde senkron olmadığını fakat Skor vb. her şeyin senkron olarak çalışmaya devam ettiğini, dolayısıyla bir yarış kondisyonunun ortaya çıktığını gösteriyor.



Server sol ekran, Client sağ ekran olmak üzere; gördüğünüz resimde Client'in mavi masasının

efektleri yarış kondisyonundan ötürü senkron değil. Fakat yukarıda görünen skor tablosu senkron.

Bütün Networking sorunlarını çözdükten sonra ise oyunumuza bir tur takibi eklememiz gerekiyor. Bu Server'ın sadece X hamlesi yapabilmesini ve Client'in O hamlesi yapmasını beklemesini gerektiriyor. Bütün komutlar Server üzerinden çalıştığı için de öncelikle bir ilkel değişken ile bu tur sayacını senkron hale getirmemiz gerekiyor.

Oyun artık 2 cihaz üzerinden aynı ağ içerisinde çok oyunculu halde oynanabildikten sonra, Menü, ses efektleri, müzik, skor kaydı, başarımlar, yayınlama gibi bir çok küçük ama bir okadar da oyunu oyun yapan şeylere sıra geliyor.

Vize dönemi ve sonrası için planlar

Vizeye kadar yaptığımız geliştirmeler bunlardan ibaret. Daha sonrasında ise hala tek oyunculu pratik modu için çok düşük kademeli bir Minimax algoritması eklemek istiyoruz. 0 kademede rastgele, 1 kademede puan kazanmak adına ve 3 kademede ise rakibin puan almasını engelleyecek şekilde programlamayı düşünüyoruz. Oluşturduğumuz listelerden ötürü de en orta karenin 4, ortak karelerin 2 ve diğer köşe karelerin de 1 ağırlıkta olduğunu biliyoruz. Minimax için arkada çalışan bir prosedüre ihtiyacımız var. Bu Network üzerinde senkron olmayacağı için Mirror ile entegre çalışmayacak.

Oyuna aynı zamanda bir lobi sistemi ekleyip yanında da eşleştirme, hazır olma, lobiden çıkma gibi temel özellikleri eklemeyi düşünüyoruz. Bu lobi sistemi 2 oyuncunun oyun içerisinde birbirini bulup oyunlarına katılabilmelerini sağlayacak.

Daha daha gelecek planlarımızı paylaşmak gerekirse; tez projesine yetişmesi neredeyse imkansız olsa da, oyunu aynı zamanda Google ve Valve üzerinden onaylatıp yayınlamayı, gerekli hesap takiplerini bulutta yapmayı, server üzerinden ortak lobiler oluşturmayı ve dağıtıcı platform üzerinden oyun içindeki triggerlar ile başarımlar sistemi eklemeyi düşünüyoruz. Fakat bu bahsedilenler bizim tez konumuzun problemine dahil değil.

KANAKÇA

1. https://docs.unity3d.com/Manual/
2. https://mirror-networking.com/docs/
3. https://en.wikipedia.org/wiki/Minimax
4. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
5. https://www.youtube.com/user/Brackeys
6. https://www.youtube.com/c/FatDino
7. https://en.wikipedia.org/wiki/Tic-tac-toe
8. https://www.youtube.com/channel/UCGIF1XekJqHYIafvE7l0c2A
9. https://docs.microsoft.com/tr-tr/dotnet/csharp/
10. https://docs.unity3d.com/Packages/com.unity.render-pipelines.lightweight@5.10/manual/index.html
11. KAAN GÖKCESU(2017), Online minimax optimal density estimation and anomaly detection in nonstationary environments
12. AMIN FARIDYAHYAEI(2017), A multi-level continuous minimax location problem with regional demand
13. Dzhafarov Vakif, Karamançıoğlu A, Çetintaş S (1997),Minimax optimal control for one class of uncertain systems