AĞ PROGRAMLAMA DERSİ PROJE RAPORU

Karadeniz Teknik Üniversitesi

Bilgisayar Mühendisliği Bölümü

ÇOK OYUNCULU WEBSOCKET OYUNU

MASTERMIND SAYI OYUNU

HAZIRLAYANLAR:

Halil İbrahim ÇAKIR- [425474]

Mahsum Taha KORKMAZ- [412746]

Hakan TEMİRKAYNAK- [425430]

DERSİ VEREN ÖĞRETİM ÜYESİ:

Öğr. Gör. Dr. ZAFER YAVUZ

"Mastermind Multiplayer": Ağ Programlama Dersi Proje Raporu

1. Giriş

Bu bölüm, projenin genel vizyonunu ve Ağ Programlama dersi kapsamında neden önemli bir pratik çalışma olduğunu ortaya koymaktadır. Projenin teknik hedefleri, modern web uygulamalarının karşılaştığı temel bir soruna, yani anlık ve çift yönlü iletişim ihtiyacına nasıl odaklandığını açıklamaktadır.

1.1. Projenin Amacı ve Kapsamı

Bu projenin temel amacı, klasik Mastermind oyunundan esinlenerek geliştirilen bir sayı tahmin oyununu, modern ağ teknolojilerinden WebSocket kullanarak çok oyunculu ve gerçek zamanlı bir dijital platforma dönüştürmektir. Proje, oyuncuların 100 ile 999 arasında rastgele belirlenen 3 basamaklı bir sayıyı, doğru rakam ve doğru pozisyon (🟢-Yeşil) ile doğru rakam fakat yanlış pozisyon (🟡-Sarı) ipuçlarına dayanarak tahmin etmeye çalıştığı rekabetçi bir ortam sunar. Bu çalışma, sadece bir oyun geliştirmenin ötesinde, günümüzün sohbet uygulamaları, canlı veri panelleri ve interaktif oyun platformları gibi anlık iletişim gerektiren sistemlerin temelini oluşturan mimari prensipleri uygulamalı olarak keşfetmeyi hedeflemektedir.Geleneksel istek-cevap (request-response) modeline dayalı HTTP protokolünün yetersiz kaldığı senaryolara odaklanılmıştır. Proje kapsamında geliştirilen sistem, sunucunun istemcilere proaktif olarak veri gönderebildiği, tüm oyuncuların oyun durumunu (örneğin skor tablosu, geri sayım) eş zamanlı olarak görebildiği, düşük gecikmeli bir deneyim sunmaktadır. Bu rapor, projenin bu hedeflere ulaşmak için tasarlanan teknik mimarisini ve altında yatan ağ programlama mantığını detaylı bir şekilde inceleyecektir.

2. Sistem Mimarisi

Bu bölümde, projenin iskeletini oluşturan istemci-sunucu mimarisi ve bu mimari için yapılan kritik teknoloji seçimlerinin stratejik gerekçeleri açıklanmaktadır. Doğru mimari ve teknoloji seçimi, projenin performans, ölçeklenebilirlik ve gerçek zamanlılık hedeflerine ulaşmasında belirleyici bir rol oynamıştır.

2.1. İstemci-Sunucu Modeli ve Teknoloji Seçimi

Sistemin mimarisi, ağ programlamanın temelini oluşturan klasik istemci-sunucu modeline dayanmaktadır. Bu model, sorumlulukların net bir şekilde ayrılmasını sağlamıştır:

1. **Sunucu (server.js):** Node.js ortamında çalışan ve projenin beyni olarak görev yapan merkezi bir uygulamadır. Sunucunun iki temel sorumluluğu bulunmaktadır:
2. **HTTP Sunucusu:** Başlangıçta istemciye ait statik dosyaları (client.html, client.js) tarayıcıya sunar.
3. **WebSocket Sunucusu:** ws kütüphanesi aracılığıyla kurulan bu katman, tüm oyun mantığını, oyuncu durumlarını, skorlamayı ve en önemlisi istemciler arası gerçek zamanlı iletişimi yönetir.
4. **İstemci (client.html , client.js):** Herhangi bir modern web tarayıcısında çalışabilen, saf HTML, CSS ve JavaScript kullanılarak geliştirilmiş hafif bir arayüzdür. İstemci, sunucudan gelen verileri görselleştirir ve kullanıcı etkileşimlerini

sunucuya iletir.Bu teknoloji yığınının bilinçli olarak yalın tutulması (harici bir frontend framework kullanılmaması), projenin amacına uygun olarak, WebSocket protokolünün ve olay tabanlı ağ programlamanın temel mekaniklerine derinlemesine odaklanmamı sağlamıştır.

2.2. Protokol Tercihi: Neden HTTP Yerine WebSocket?

Projenin gerçek zamanlılık gereksinimi, iletişim protokolü seçimini doğrudan etkilemiştir. Standart HTTP, anlık ve çift yönlü veri akışı için tasarlanmamıştır. Bu nedenle, projenin temel taşı olarak WebSocket protokolü tercih edilmiştir. İki protokol arasındaki temel farklar aşağıdaki tabloda özetlenmiştir:| Özellik | Geleneksel HTTP | WebSocket || ------ | ------ | ------ || **Bağlantı Modeli** | Her istek için yeni bağlantı kurulur (istek-cevap). | Tek bir TCP bağlantısı sürekli açık kalır. || **İletişim Yönü** | Yarı çift yönlü (Half-duplex): İletişim istemci tarafından başlatılır. | Tam çift yönlü (Full-duplex): Sunucu ve istemci aynı anda veri gönderebilir. || **Gecikme (Latency)** | Yüksek (100-300ms), her istekte TCP el sıkışması (handshake) tekrarlanır. | Çok düşük (1-5ms), ek yük (overhead) minimumdur. || **Kullanım Alanı** | Web sayfaları, API sorguları gibi durum bilgisi tutmayan işlemler. | Gerçek zamanlı oyunlar, sohbet uygulamaları, canlı veri akışları. |
WebSocket'in sağladığı tam çift yönlü (full-duplex) iletişim ve milisaniyelerle ölçülen düşük gecikme avantajı, projedeki anlık liderlik tablosu güncellemeleri, tüm oyuncular için senkronize geri sayım ve tahmin sonuçlarının anında iletilmesi gibi kritik özellikler için bir tercih değil, bir zorunluluktur. Bu protokol tercihi, sistemin temelini oluşturan olay tabanlı iletişim mimarisini ve veri akışını doğrudan şekillendirmiştir; bu yapıyı sonraki bölümde detaylandıracağız.

3. İletişim Protokolü ve Veri Akışı

Bu bölümde, sistemin dinamizmini sağlayan olay tabanlı iletişim yapısı ve bu yapı içinde akan verinin formatı incelenmektedir. İstemciler ve sunucu arasındaki etkileşimin nasıl standartlaştırıldığı ve çok oyunculu bir ortamda oyun durumunun tutarlılığının nasıl korunduğu bu bölümün ana odak noktasıdır.

3.1. Olay Tabanlı İletişim Mekanizması

Sunucu ile istemciler arasındaki tüm iletişim, WebSocket bağlantısı üzerinden gönderilen, hafif ve hem insan hem de makine tarafından kolayca okunabilen JSON formatındaki mesajlarla sağlanır. Sistem, doğası gereği olay tabanlıdır; yani sunucu, belirli oyun olayları gerçekleştiğinde (örneğin, bir round'un başlaması, bir oyuncunun doğru tahminde bulunması) tüm istemcilere proaktif olarak bilgi yayınlar (broadcast). Bu mekanizma, istemcilerin sürekli olarak sunucuyu yoklaması (polling) ihtiyacını ortadan kaldırarak ağ trafiğini ve gecikmeyi minimize eder.Temel mesaj türleri, veri akışının yönüne göre şu şekilde sınıflandırılabilir:

- **İstemciden Sunucuya Gönderilen Mesajlar:**
- setName: Oyuncunun oyuna katılırken kendi ismini belirlemesini sağlar.
- guess: Aktif round sırasında oyuncunun 3 basamaklı sayı tahminini sunucuya iletir.
- **Sunucudan İstemciye Gönderilen (Yayın) Mesajlar:**
- playerJoined / playerLeft: Bir oyuncu katıldığında veya ayrıldığında tüm istemcilere bildirir.

- matchStart / roundStart: Yeni bir maçın veya round'un başladığını ve ilgili bilgileri (round numarası vb.) tüm oyunculara iletir.
- hint: Bir oyuncunun tahminine karşılık kişiye özel ipucu (🟢-Yeşil/🟡-Sarı) gönderir.
- correctGuess: Bir oyuncu doğru tahmini yaptığında sadece kendisine özel bir onay mesajı gönderir.
- playerWonRound: Bir oyuncu round'u kazandığında, bu bilgiyi diğer tüm oyunculara duyurur.
- leaderboard: Puan durumunda bir değişiklik olduğunda güncel skor tablosunu tüm oyunculara gönderir.
- roundEnd / matchEnd: Bir round veya maç tamamlandığında, sonuçları (doğru sayı, kazananlar) tüm oyuncularla paylaşır.
- matchCountdown: Maçlar arasındaki bekleme süresinde her saniye geri sayım bilgisini yayınlar.
- error: Geçersiz bir işlem olduğunda ilgili istemciye hata mesajı gönderir.

3.2. Oyun Durumu (State) Senkronizasyonu ve Bütünlüğü

Çok oyunculu bir ortamda en kritik zorluklardan biri, tüm istemcilerin tutarlı bir oyun durumuna (game state) sahip olmasını sağlamaktır. Bu projede bu sorun, sunucuyu **"tek doğru kaynak" (single source of truth)** olarak konumlandırarak çözülmüştür. Sunucudaki gameState nesnesi, oyunun anlık fotoğrafını tutar ve isMatchActive (maçın aktif olup olmadığı), targetNumber (o anki round'un hedef sayısı), roundTimer (aktif zamanlayıcı referansı) ve bağlı olan tüm oyuncu verilerini içeren bir Map yapısı gibi kritik verileri barındırır. İstemciler kendi başlarına oyun durumunu (örneğin, kalan süre, skor) asla değiştirmezler; yalnızca sunucudan gelen leaderboard, roundStart gibi mesajlara tepki vererek kendi arayüzlerini güncellerler.Bu merkeziyetçi yaklaşım, "Yarış Durumu" (Race Condition) gibi potansiyel tutarsızlık risklerini de ortadan kaldırır. Oyun akışını belirleyen en kritik geçişler (bir round'un başlatılması veya bitirilmesi gibi) doğrudan oyuncu eylemlerine değil, tamamen sunucu tarafında çalışan otonom ve merkezi zamanlayıcılara (matchTimer, roundTimer) bağlıdır. Bu tasarım, oyuncuların ağ gecikmesi (latency) gibi değişken faktörlerden etkilenmesini tamamen ortadan kaldırır. Bir round, bir oyuncunun 'bitir' komutunu göndermesiyle değil, sunucudaki zamanlayıcının deterministik olarak endRound fonksiyonunu tetiklemesiyle sona erer. Bu, oyunun adaletini ve tutarlılığını garanti altına alır. Bu merkezi zamanlayıcıların projenin otonom yapısındaki rolünü bir sonraki bölümde daha detaylı inceleyeceğiz.

4. Otonom Döngüler ve Zaman Yönetimi

Bu bölümde, projenin bir moderatör veya insan müdahalesi olmadan sürekli olarak çalışmasını sağlayan otonom döngülerin ve merkezi zaman yönetimi mantığının, ağ trafiği ve oyuncu senkronizasyonu üzerindeki etkileri ele alınmaktadır. Bu mekanizmalar, oyunun öngörülebilir ve adil bir ritimde ilerlemesini garanti eder.

4.1. Sunucu Taraflı Zamanlayıcıların Rolü

Sunucu üzerinde çalışan iki ana otonom döngü, oyunun tüm akışını yönetir ve oyuncu aktivitesinden bağımsız olarak çalışır. Bu, oyunun kendi kendine yeten bir sistem olmasını sağlar.

- **Maç Döngüsü (45 Saniye):** MATCH_INTERVAL sabiti (45000 ms) ile kontrol edilen bu döngü, 5 round'dan (ROUNDS_PER_MATCH=5) oluşan bir maç sona erdikten 45 saniye sonra otomatik olarak yeni bir maç başlatır. Bu bekleme süresi boyunca, broadcastMatchCountdown fonksiyonu her saniye tüm istemcilere yeni maça kalan süreyi bildiren bir mesaj yayınlar. Bu düzenli yayın, bir "kalp atışı" (heartbeat) görevi görerek hem oyuncuları bilgilendirir hem de bağlantıların aktif olduğunu teyit ederken tüm istemcileri bir sonraki başlangıç için senkronize tutar.
- **Round Döngüsü (60 Saniye):** ROUND_DURATION sabiti (60000 ms) ile yönetilen bu döngü, her bir tahmin periyodunu 60 saniye ile sınırlar. Süre dolduğunda, zamanlayıcı otomatik olarak endRound fonksiyonunu tetikler. Bu sayede, hiçbir oyuncu doğru tahmini yapamasa bile round'un tüm oyuncular için tam olarak aynı anda sona ermesi garanti altına alınır. Bir round bittikten 5 saniye sonra, eğer maç tamamlanmamışsa, bir sonraki round otomatik olarak başlar.

4.2. Ağ Trafiği ve Senkronizasyon Üzerindeki Etkileri

Bu sunucu taraflı zamanlayıcılar, oyuncu sayısı veya aktivitesinden bağımsız olarak düzenli ve öngörülebilir bir ağ trafiği yaratır. Örneğin, maçlar arasındaki 45 saniyelik bekleme süresince, broadcastMatchCountdown fonksiyonu her saniye sunucu saatine dayalı geri sayım bilgisini içeren bir matchCountdown mesajı yayınlar. Bu sadece bir bilgilendirme değil, aynı zamanda tüm istemcilerin saatlerini bir sonraki başlangıç anına senkronize eden bir 'kalp atışı' (heartbeat) işlevi görür.En önemlisi, bu merkezi zamanlama mantığı, tüm oyuncuların oyun deneyimini mükemmel bir şekilde senkronize eder. Bir round'un veya maçın başlaması ve bitmesi gibi kritik anların her oyuncu için aynı anda gerçekleşmesi, tamamen sunucu saatine dayalı bu otonom yapı sayesinde sağlanır. Bu, ağ gecikmelerinden kaynaklanabilecek haksız avantaj veya dezavantajları ortadan kaldırarak adil bir oyun ortamı yaratır. Projenin teknik analizini tamamlayarak, elde edilen kazanımları ve sonuçları değerlendireceğimiz son bölüme geçebiliriz.

5. Sonuç ve Kazanımlar

Bu son bölümde, projenin tamamlanmasıyla Ağ Programlama dersi kapsamında edinilen teorik bilgilerin pratiğe dökülmesiyle elde edilen teknik bilgi birikimi ve deneyimler özetlenmektedir. Proje, sadece bir ödev olmanın ötesinde, modern dağıtık sistemlerin temel dinamiklerini anlamak için değerli bir uygulama sahası sunmuştur.

5.1. Projenin Değerlendirmesi ve Öğrenimler

"Mastermind Multiplayer" projesi, Ağ Programlama dersinde öğrenilen teorik kavramları somut bir ürün üzerinde uygulama açısından son derece başarılı olmuştur. Proje süresince, gerçek zamanlı ve çok kullanıcılı bir sistemin tasarımından implementasyonuna kadar olan

süreçte karşılaşılan zorluklar, önemli öğrenimler sağlamıştır.Bu proje sayesinde edindiğim temel kazanımlar şunlardır:

- **WebSocket Protokolü:** HTTP'den temel farklarını ve sürekli açık bir TCP bağlantısı üzerinden tam çift yönlü iletişimin nasıl kurulduğunu pratik olarak deneyimleme imkanı buldum. Düşük gecikmenin interaktif uygulamalar için ne kadar kritik olduğunu bizzat gözlemledim.
- **TCP Bağlantı Yönetimi:** WebSocket'in TCP üzerine kurulu yapısı sayesinde, her bir mesaj için yeniden bağlantı kurma (handshake) yükünden kurtularak verimli bir iletişim kanalının nasıl yönetildiğini öğrendim.
- **Dağıtık Sistemlerde Durum Yönetimi:** Çok sayıda istemcinin durumunu (state) sunucu merkezli bir mimari ile senkronize tutmanın zorluklarını ve broadcast gibi yayın mekanizmalarının bu soruna nasıl zarif bir çözüm getirdiğini anladım.
- **Otonom Sistem Tasarımı:** Sunucu tarafı zamanlayıcılar kullanarak, kullanıcı etkileşiminden bağımsız, kendi kendine işleyen, tutarlı ve öngörülebilir bir oyun döngüsü tasarlamanın önemini kavradım.
- **Mastermind Algoritması ve Oyun Mantığı:** Tahminleri (checkGuess fonksiyonu ile) değerlendiren, ipuçlarını (🟢-Yeşil/🟡-Sarı) hesaplayan ve puana etki eden zaman ve tahmin sayısı gibi faktörlere dayalı karmaşık bir skorlama (calculateScore fonksiyonu) mantığını sunucu tarafında hatasız bir şekilde tasarlama ve uygulama becerisi kazandım.Sonuç olarak, bu proje ağ programlamanın sadece veri paketleri göndermekten ibaret olmadığını; aynı zamanda durum yönetimi, senkronizasyon ve dağıtık sistem mimarisi gibi karmaşık konuları da içerdiğini derinlemesine anlamamı sağlamıştır.