PROGRAMLAMA LABORATUVARI PROJE 1

1st Tolga Ferhan Küçük
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
İzmit/Kocaeli
tolgaferhankucuk@hotmail.com

2nd Ahmet Alp Akay
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
İzmit/Kocaeli
ahmeta3557@gmail.com

I. ÖZET

Yeni prolab projemiz olan otonom hazine avcısında amaçlamak istediğimiz şey anasayfa girişinden istenilen boyut yazıldığında o boyutun karesi kadar büyüklüğünde bir alan oluşturmak ve bu alanda oyunumuz olan hazine avcısı oyununu başlatabilmek. Bu oyunda bulunan nesnelerimiz hareketsiz engeller (dağ,duvar,ağaç,kaya), hareketli engeller(kuş,arı), karakterimiz ve hazinelerden oluşuyor.

Bu nesneler hazineler hariç haritada yer kaplıyorlar haritada yer kaplamalarının amacı nesnelerin birbirine çarparak engel oluşturması olacaktır. Bu olayın nasıl gerçekleştiği giriş ve yöntem kısımlarında detaylı açıklanacaktır.

Asıl bu oyunun amacı ise bu engellerden geçip hazineleri en kısa yol problemine göre hesaplayıp en kısa yoldan hazineleri toplamasını amaçlamaktır. Fakat bunu büyük bir haritada ve üstelik sisli bir ortamda nasıl yapacağımızı düşündüğümüzde internetten de araştırdığımız A* algoritmasının bizim yapacağımız bu projeye tam uygun olduğunu düşündük.Bu algoritmanın nasıl işlediği konusunda detaylı işleyiş giriş ve yöntem kısmında yazılacaktır.

II. GİRİŞ

Projemizde kullandığımız görselleştirme yöntemi Unity olup programlama dilimiz ise csharp. Hazır kütüphaneler kullanmaktan kaçınıp kuyruk yapısı ,kısa yol algoritmasını kendimiz yapıp en kısa yol hesaplamasını kendi scriptimiz ile yaptık. Projede ilk Anasayfa bizi karşılıyor burada istenilen boyut girildiği zaman o boyut yeni int değere atanarak int değeri karesiyle çarpılarak yeni karesel ızgaramız oluşturuluyor ve Anasayfa dan oyun ekranına geçilerek oyun başlamış oluyor. Öncelikle engeller sınıfı oluşturduk bu engelleri efektif bir şekilde kullanabilmemiz için bu engelleri bir objeye dönüştürmemiz gerekti unitynin gameobject fonksiyonunu kullanamayacığımız için biz de kendi object classımızı oluşturup buradan her kullanacağımız objeye id ve kaplayacağı boyutu oyuncunun idsini hem constructor hem de get set metotlarıyla sarmalladık ve objeleriimizi bu şekilde oluşturduk. Sonrasında oluşturduğumuz bu objelerden ilk engellerin işlevi için engeller sınıfı oluşturup bu sınıfta çeşitli metotlarla hareketsiz olanlar ve hareketli olanlar için kapladığı alan ve ayrıca hareketli engeller için gerekli metotlarla hareket işlevleri kazandırdık. Bu tarz metotları işledik bunların detayları yöntem kısmında verilecektir. Engellerin işlevleri bittiğine göre sıra karakterin ve hazinelerin oluşumuna geliyor.Öncelikle karakter ve hazineler zaten object sınıfında nesne haline gelmişti. Öncelikle karakterden başlanarak sırasıyla sandıklar, ağaçlar, taşlar, duvar, dağ ve haraeketli engeller oyunun dışına taşmayacak ve birbirleriyle çakışmayacak şekilde kontrol ediliyor ve yerleri belirleniyor. ID'leri de bu kısımda belirleniyor ve object classında tutuluyor. Bu kontrol yapılırken her nesnenin boyutu birbirinden farklı olduğu için object sınıfında tutulan boyut değişkenlerine göre kontrol sağlanıyor. Bütün bu kontroller sağlanıp bütün nesneler oluşturulduktan sonra görsel olarak da oluşturulması gerekiyor. DrawField() fonksiyonu ile yine her nesnenin kendi boyutuna göre merkez noktaları manipüle edilerek önceden atanmış prefablar oyun alanında gerçekleştiriliyor. Oyun alanındaki diğer UI elemanları da yerleştiriliyor. Bu elemanlardan olan yeniden yükle butonu bütün sahneyi yeniden yükleyerek haritanın baştan oluşturulmasını sağlıyor. Sis butonu ise sisin açılmasını sağlıyor ve ardından karakter hareket etmeye başlıyor.

III. YÖNTEM

A. Sandıkların Toplanmasındaki Algoritma

Sandıkları en kısa yoldan toplamak için algoritma olarak AStar algoritmasını kullandık. Bunu kullannmamızdaki amaç internetten araştırmalarımıza göre oyunlarda veya kısa yol problemlerinde bu algoritmanın sıklıkla kullanılmasıdır bu algoritmanın açıklaması şu şekildedir. Bu algoritma bir başlangıç

düğümünden bir hedef düğüme giderken bir yol bulmaya çalışır. İlk olarak başlangıç ve varış noktalarını alır. Başlangıç noktasından başlayarak komşu noktaların gcost(kaç adımda o noktaya varıldığı), hcost(varış noktasına ne kadar uzaklıkta olduğu) ve ikisinin toplamı olan fcost değerlerini karşılaştırır ve düşük olan noktaları seçer. En sonunda varış noktasına ulaşıldığında seçilmiş olan noktaları kuyruk yapısı şeklinde döndürür. A* algoritması , her bir hedef noktasından diğerine

gitmek için çalıştırılır. Bu hedef noktaları görülen sandıklar öncelikli olacak şekilde sisli alanları en optimum şekilde görecek şekilde oyun oynanmaya başlamadan önce belirlenir. Bu noktalarda herhangibir engel bulunuyorsa hedef noktası kendine en yakın boş noktayla değiştirilir ve a* algoritması o nokta için uygulanır. Heuristic fonksiyonu,

her bir düğümün hedefe olan tahmini maliyetini hesaplar. Bu, Manhattan mesafesi (düz hat mesafesi) kullanılarak yapılır.GetNeighbours fonksiyonu, bir düğümün komşularını belirler. Bu, ızgara üzerinde yukarı, aşağı, sağa ve sola olan komşuları döndürür.ReconstructPath fonksiyonu, başlangıçtan hedefe kadar olan en kısa yolu yeniden oluşturur. Bu, hedef düğümden başlayarak geriye doğru hareket eder ve her adımda en düşük maliyetli komşuyu seçer.Son olarak, GetLowestNeighbor fonksiyonu, bir düğümün en düşük maliyetli komşusunu belirler.Ve kod bu şekilde işler.

B. Harita ve Nesnelerin Oluşturulması ve Karakterin Algılaması

gameControl1 classımızda oyuncumuzu yolumuzu ve engellerimizi objelerini tanımladık.Aynı zamanda yolumuzu kuyruk adında bir list e atayarak yolumuzu tutabilmeyi amaçladık. Awake fonksiyonuyla yolumuzu bir matris gibi

düşündüğümüzde harita alanının tümünü çeşitli for döngüleri ile -1 olarak atadık. Yani yolumuz -1 olarak atanmış oldu.Objelerimizin rastgele dağılabilmesi ve karakterimizin rastgele bir noktada oluşabilmesi için randomKonum fonksiyonu oluşturduk.Bu fonksiyon oluşturduğumuz nesnelerin id lerini ve int türündeki sayac alır.Burada çeşitli dögüler ile kaç tane engel oluşturulacağı ve sandıklardan kaç tane oluşacağı yazılmıştır. kontrol metodu, belirli bir konum için oyun alanında nesnelerin yerleştirilip yerleştirilemeyeceğini kontrol eder. Bu metot, verilen konumda belirtilen boyutta bir nesnenin yerleştirilebilir olup olmadığını belirlemek için kullanılır. Eğer belirtilen konumda bir nesne varsa veya oyun alanının dışına çıkılıyorsa, false döndürülür ve nesne yerleştirilemez. Aksi takdirde, true döndürülür ve nesne başarıyla yerleştirilebilir. drawField metodu, oyun alanını belirli nesnelerle doldurmak

için kullanılır. Bu metot, oyunun başlangıcında veya herhangi bir değişiklikte, oyun alanını görsel olarak oluşturmak için kullanılır. Bu sayede, oyun sahnesindeki nesnelerin yerleştirilmesi ve düzenlenmesi sağlanır. drawField metodu genellikle oyunun başlangıcında çağrılır ve oyun alanındaki nesnelerin yerleştirilmesini sağlar.Bu nesneler hareketli ve hareketsiz engellerimiz,sandıklar ve playerımızdır. Bunları switch case yapısıyla oluşturup her oluşturduğumuz nesnenin id si ile çağırıyor ve oluşturuyoruz.

C. Move Class

Awake(): Bu metot, oyun nesnesi oluşturulduğunda çağrılır. Karakterin başlangıç pozisyonunu belirler, otomatik hareket metodunu tekrar eden bir döngü başlatır ve en kısa yol algoritmasını kullanarak bir hedefe doğru hareket yolunu hesaplar.Bu algoritmada A* algoritmasını tekrar kullanmış oluyoruz.

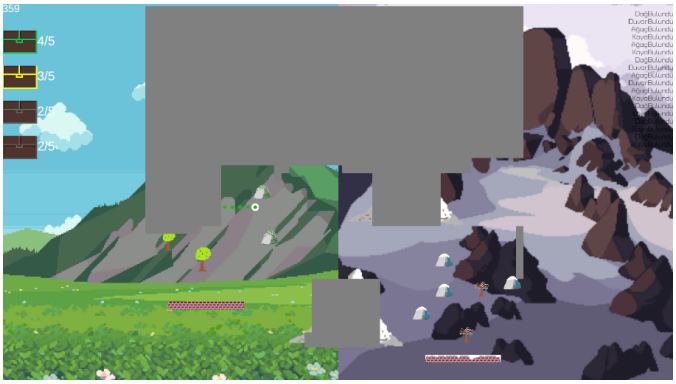
Update(): Her karede bir kez çağrılır. Karakterin hareketini kontrol eder, karakterin en kısa yol takip etmesini sağladık.

MoveCharacter(): Karakterin en kısa yolu takip ederek hareket etmesini sağlar. oluşturduğumuz Yol listesindeki her bir konuma doğru hareket eder.

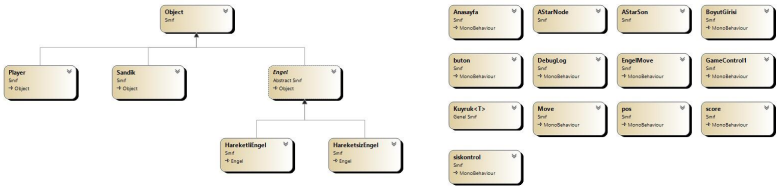
OnTriggerStay2D(Collider2D other): Karakterin temas ettiği nesneleri kontrol etmek için bu metot kullanılır. Sandık ve yoldan temas aldığında ilgili işlemleri gerçekleştirir.

OtoHareket(): Otomatik hareket için kullandığımız bir metot.Bu metot A* herustic ve manahtan algoritmalarını kullanır.

IV. DENEYSEL SONUÇLAR



V. UML DİYAGRAMI



VI. SONUÇ

CSharp programlama dili kullanılarak karakterin otomatik hareket eden bir oyunun geliştirilmesini amaçladık.

Oyunun İşlevselliği: Oyun, rastgele oluşturulan haritalar üzerinde karakterin en kısa yolu takip ederek tüm hazine sandıklarını toplamasını sağlayacak şekilde tasarladık. Karakter, engellere takılmadan ve en kısa sürede hedefine ulaşabilmektedir. Ayrıca, oyunun performansı yeterli seviyededir ve haritaların rastgele oluşturulması çeşitli algoritmalar kullanılarak gerçekleştirdik.

Kullanıcı Arayüzü ve Görsellik: Oyunun arayüzü net ve kullanıcıodaklı şekilde tasarlamayı amaçladık. Harita oluşturma ve başlatma işlemleri için kullanıcıya kolaylık sağlayan butonlar ekledikr. Oyun sırasında karakterin hareketi ve bulunan hazine sandıkları kullanıcıya görsel olarak aktardık.Bunları yaparken windows formu kullandık.

Veri Yapıları ve Nesne Yönelimli Programlama : Projede kullanılan veri yapıları ve nesne yönelimli programlama prensipleri bir kullandık. Sınıflar ve metodlar, projenin gereksinimlerine uygun olarak tasarladık ve deneme classları oluştursak dahi bunları bir noktada kullanmayı amaçladık. Encapsulation, Inheritance, Polymorphism, Abstraction gibi prensipleri olabildiğince kullandık.

Algoritma: En kısa yol algoritması olarak A* herustic ve manahtan algoritmalarını kullandık, karakterin hazine sandıklarını en kısa sürede toplamasını sağlamak için etkili bir şekilde tasarlamaya çalıştık. Ayrıca, harita oluşturma işlemi için kullanılan algoritma, rastgele üretilmesini sağladık.

VII. KAYNAKÇA

-wikipedia
-stackoverflow
-Microsoft Learn
-github
-W3school
-BTK akademi
-coderspace
-medium.com
-bilgisayarkavramlari.com
-DonanımHaber Forum
-https://www.youtube.com/watch?v=ScOXXLPA1kQ