Programlama Laboratuvarı-2 1. Proje

Programming Laboratory-2 1. Project

1. Eyüp Ensar Kara
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
eyupensarkara0@gmail.com

2. Yunus Hanifi Öztürk
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
oyunushanifi@gmail.com

***Özetçe*—Bu belge Kocaeli üniversitesi bilgisayar mühendisliği bölümü programlama laboratuvarı-2 1. Proje ödevi için hazırlanmıştır.**

Anahtar Kelimeler — C#, Windows Forms, 2d Game, Path Finding, A* Algorithm

I. Özet

Projenin temel amacı, nesneye yöneliik yapıları kullanarak iki boyutlu bir düzlemde oyun oluşturup bu oyunu en kısa yoldan bitiren bir algoritma yazmaktır.

II. Giriş

Otonom Hazine Avcısı projesi, nesneye yönelik programlama ve veri yapıları bilgilerinin pekiştirilmesi ve uygulanmasını hedefleyen kapsamlı bir yazılım geliştirme çalışmasıdır. Bu projede, otonom hareket eden bir karakterin, rastgele oluşturulmuş bir harita üzerinde bulunan çeşitli hazineleri toplaması amaçlanmaktadır. Aynı zamanda, karakterin engellere takılmadan, en kısa sürede ve en kısa yoldan tüm hazine sandıklarını toplamasını sağlayacak bir algoritmanın tasarlanması gerekmektedir.

Bu projenin temel amacı, öğrencilerin nesneye yönelik programlama ve veri yapıları bilgisini pekiştirmek, uygulamak ve problem çözme becerilerini geliştirmektir. Bu kapsamda, C++, C# veya Java gibi programlama dilleri kullanılarak yazılım geliştirilecek ve projenin her aşamasında veri yapıları ve algoritmaların etkin bir şekilde kullanılması sağlanacaktır.

Projede, hareketli ve hareketsiz olmak üzere iki tür engel bulunmaktadır. Hareketsiz engeller arasında ağaçlar, kayalar, duvarlar ve dağlar yer alırken, hareketli engeller arasında kuşlar ve arılar bulunmaktadır. Bu engeller, otonom karakterin ilerlemesini zorlaştırarak algoritmanın stratejik kararlar almasını gerektirir. Ağaçlar genellikle ormanlık alanlarda ya da tek başlarına bulunurken, kayalar farklı boyutlarda ve şekillerde olabilir. Duvarlar yan yana gelmeyecek şekilde yerleştirilirken, dağlar genellikle dağ kümeleri oluşturarak engel oluşturur. Hareketli engeller, belirli bir mesafede yukarı-aşağı veya sağ-sol yönde hareket ederek karakterin yolunu keser. Bu engellerin çeşitliliği, otonom karakterin çevresini analiz etme ve stratejik bir rota oluşturma yeteneğini geliştirir. Bu sayede, karakter, engelleri aşmak için akıllıca hareket edebilir ve hedefe en kısa yoldan ulaşabilir. Engellerin bu dinamik yapısı, oyunun zorluk seviyesini artırarak otonom karakterin daha karmaşık bir oyun stratejisi geliştirmesine olanak sağlar.

III. YÖNTEM

Öncelikle oyun için gerekli sınıfları tanımlamakla başladık. Başlangıçta oyunun temelini oluşturan birçok bilginin tutulacağı temel sınıfları oluşturmakla başladık. Location ve Quad sınıflarını oluşturduktan sonra Map sınıfını oluştururken Quad dizileri kullandık. Bu sayede haritanın her bir karesine 2 boyutlu olarak istediğimiz şekilde özgürce erişilebilir yaptık.

Temel haritayı oluşturduktan sonra Barrier sınıfını tanımlayarak bu Barrier sınıfından engellerimiz olacak bir çok sınıfı türettik. Bu sınıflar içerisinde hareket edebilenleri DynamicBarrier ara sınıfını kullanarak tekrar türettik. Böylelikle Static ve Dynamic engeller arasındaki farkları kolaylıkla ayırabildik. Örneğin Dynamic engellerin hareket edilebilir olduğunu kendi içerisinde metod kullanarak override ettik. IBarrier interfacesini de tanımlayarak tek listede bütün sınıfları tutabilmeyi amaçladık. Interface sayesinde bu sınıfların metodlarını da kullanabildik.

Hareket eden nesneleri Form da göstermek için timer kullanarak picturebox'un paint işlevini tekrar tekrar çağırdık. Bu sayade hem hareketli engellerin hareketini görselleştirdik hemde haritanın 2 boyutlu bir görünümünü sürekli güncelledik.

Haritayı oluşturmak için ise random fonksiyonunu kullanarak sürekli haritada rastgele koordinatları denetledik. Engelin oluşabilmesi için hem başka bir engelle çakışmaması gerektiğini ve aynı zamanda harita sınırında oluşmamasını sağladık. Bu oluşum sürecinde aynı zamanda engelin doğru mevsimde doğru nesneyi oluşturmasını sağladık. Karakterin görüş alanı dışındaki yani başlangıç noktasının dışındaki alanı sisle kapladık. Bu özelliği oluşturmak için temel haritamız olan quads 'a bir Boolean değer vererek sis olup olmadığını ölçtük.

Karakterin oluşumunu da yaptıktan sonra otonom hareket için bir algoritma geliştirmeye koyulduk. İlk başlarda sürekli gitmediği yönlere yönelim olarak tasarlamıştık ama bu hali sürekli kendi içinde döngüye giriyordu. Sonrasında bunu önceliklendirerek doğru yöne gitmesini sağlamayı düşündük. Bu algoritma daha tutarlı çalıştı ama sonuçta engel sayısı arttıkça sürekli bir bölgede dönmeye başladı. Bundan sonra bu tip zorlama algoritmalarını kullanmaktansa en efektif olarak gördüğümüz A* algoritmasını kodumuza entegre etmeye koyulduk. Bu noktada birçok sıkıntı da yaşadık. Bu nedenle bazı sınıf yapılarımızı kalıcı olarak değiştirmek zorunda kaldık. En sonunda bir miktar da A* algoritmasına optimizasyon yaptıktan sonra kodumuzu yaklaşık 1000 boyutundaki haritalarda sorunsuz çalışacak şekilde ayarladık.

IV. Sonuç

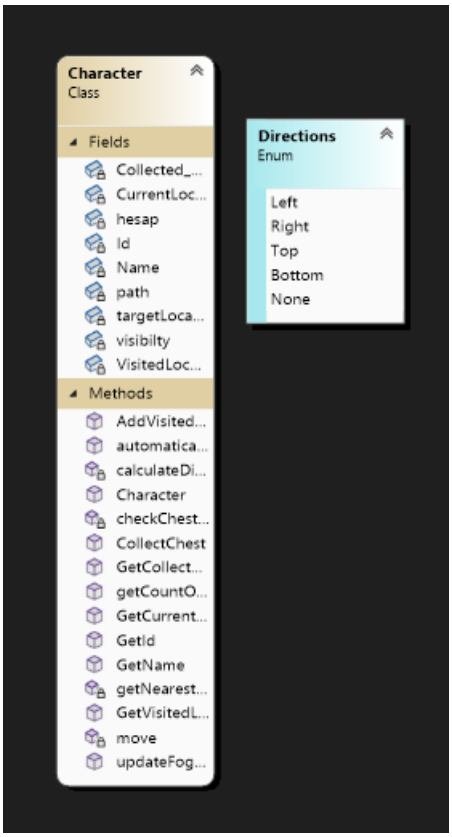
Sonuç olarak C# dilinde Windows formları oluşturabilmeyi,

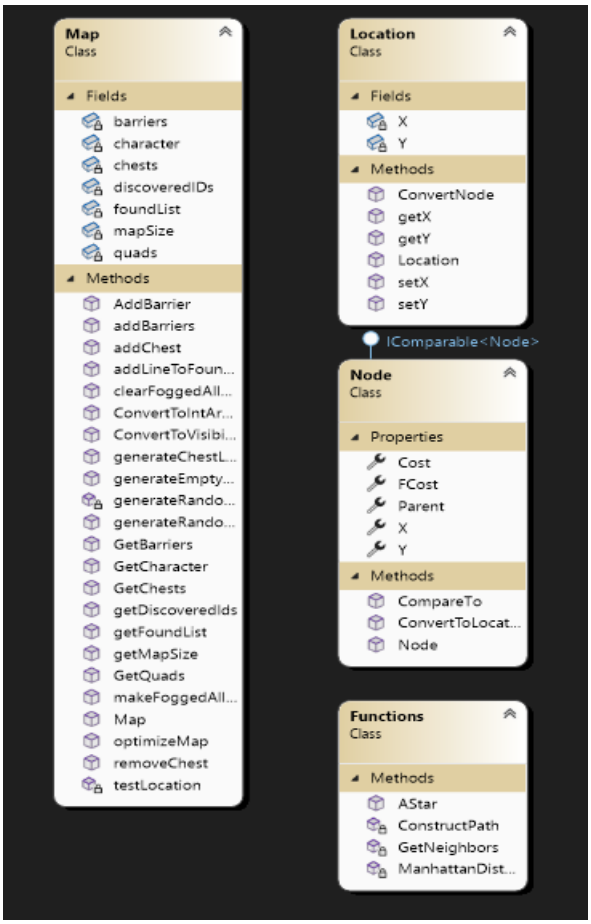
Sınıf içinde sınıf dizisi tutarak hem 2 boyutlu bir harita oluşturmayı hemde bu 2 boyutlu haritada dinamik işlem yapabilmeyi, A* algoritmasının temellerini ve Interface kullanarak bir çok yapıyı bir arada tutabilmeyi öğrendik.

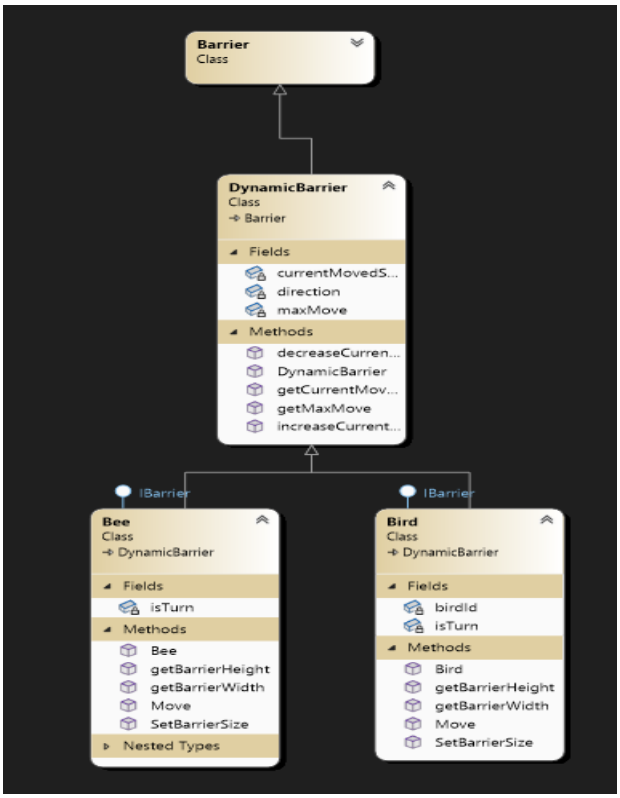
V. Katkılar

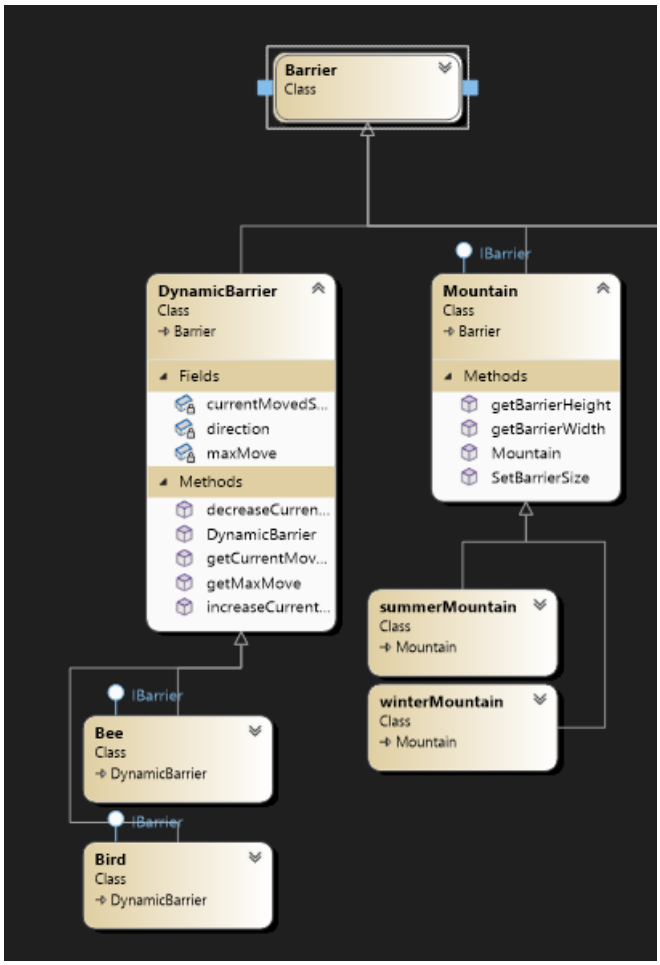
Bu projede ikimiz de hem class tanımlamak olsun hem form tasarımı olsun kendi aramızda görevlere bölmeden birlikte çalışarak yaptık.

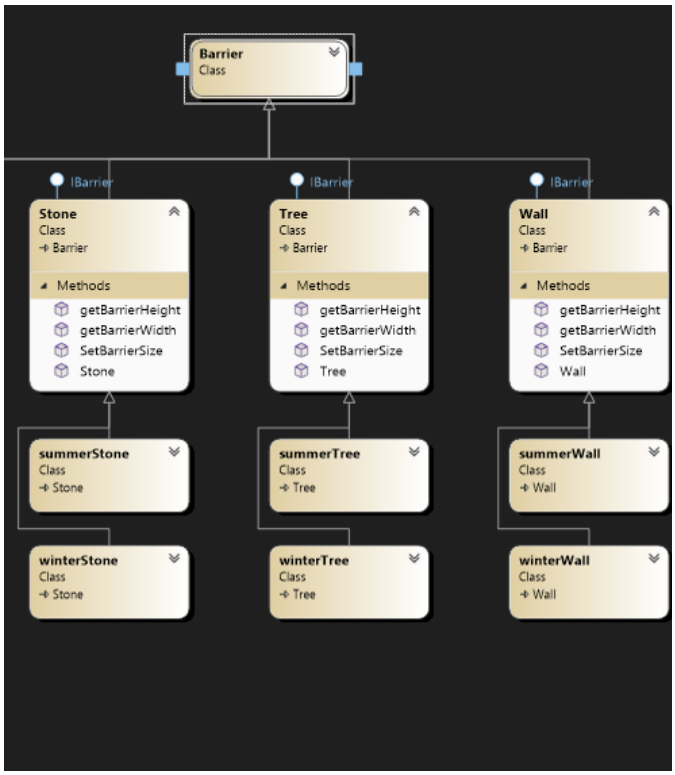
VI. UML Diagramları











Kaynakça

[1] https://stackoverflow.com/questions/2138642/how-to-implement-an-a-algorithm

[2] https://stackoverflow.com/questions/353126/c-sharp-multiple-generic-types-in-one-list

[3]https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/inheritance