ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM 4061-R PROJE RAPORU

Unity ile İzometrik ARPG Oyun geliştirme

Şuayip Deniz Talha TOPAY(18290059)

Aljoud MHD ALİ(18290992)

Doç.Dr. Gazi Erkan BOSTANCI

2 Ocak 2021

ÖZET

Bu proje kapsamında Unity üzerinde 2 dönem boyunca Isometric ARPG (Isometric Action Role Playing Game) türünde bir oyun geliştirmekteyiz. Bu rapor ise şu ana kadar geldiğimiz noktayı anlatmaktadır. Unity motorunu tercih etmemizin sebeplerinden bağzıları scripting dili olarak C# kullanması, yeni başlayanlar için kullanması kolay ancak ustalaşıldığında yeterli olması, internette fazlaca dökümanı bulunması sayılabilir. Aklımızdaki en yakın ikinci seçenek olan Unreal Engine grafik kalitesi olarak daha önde olsa da, Unity dökümanın fazlalığı artısı ve son zamanlarda grafik motoru üzerine yaptığı iyileştirmelerle birinci tercihimiz olmakta. İkisi de ücretsiz ve güçlü oyun yapım motorlarıdır.

Isometric ARPG'deki Izometrik kelimesi oyunumuzun kamera açısına gönderme yapmaktadır. Kamera açımız izometrik bir şekilde ana karakterin tepesinden bakmaktadır. Aksiyon kelimesi ise RPG (Role Playing Game)oyun türününün bir alt türü olduğunu ifade etmektedir. Klasik RPG'lerde oynanış elementleri kısıtlı tutulup daha çok rol yapma ve karakter gelişimi üzerine gidilmektedir. ARPG türünde ise oynanış daha önemli olup oyuncunun içinde bulunduğu dünya ile daha fazla etkileşime geçebilmesini istiyoruz. Klasik RPG'lerde olduğu gibi sadece düşmanın üzerine tıklayıp ölmesini beklemek yerine daha interaktif bir savaş sistemi hedeflemekteyiz. RPG ise Rol Yapma Oyunu anlamına gelmekte. İçerisinde karakter gelişim öğeleri barındıran ve kendimizden farklı bir karakteri kontrol edip olayları onun gözünden gördüğümüz ve onun yerine kararlar alabildiğimiz tarzda oyunları ifade etmektedir.

İÇİNDEKİLER

1. GİRİŞ

Halihazırda piyasada birden fazla başarılı örneği bulunan ARPG türünde kendi oyunumuzu ön plana çıkaracak bazı fikirlerimiz ve hedeflerimiz bulunmakta. Bunlardan en önemlisi akıcı ve iyi hissettiren bir gameplay loop (oyun döngüsü). Oyun döngüsü kavramı oyuncunun sürekli farklı şekillerde tekrarladığı ve oyunu yeniden oynamasını istetecek ana elementtir. Bu döngü ne kadar ilgi çekici ve tekrarlaması keyifli olursa oyuncu da o kadar oyunu oynamak isteyecek ve başarılı bulacaktır.

Fikirlerimizden ikincisi ise kontrol ettiğimiz karakter ile alakalı. Bu türün klasik örneklerinde karakter gelişimi hep karakterin üzerine giydiği eşyalar ve seviyeler üzerinden yapılmıştır. Bizim bu döngüye katmak istediğimiz şey ise karakter ile beraber kılıcının da gelişmesi. Efsanevi bir kılıca sahip olacak ana karakterimizin oyunu oynadıkça hem kendi hem de kılıcı güçlenecek ve kılıcın güçlenmesi ile beraber yeni yetenekler elde edecektir. Oyunumuzun hikayesi de bununla paralel olarak kılıcı elde etmesi ile başlayacak, bu kılıçla beraber savaşarak onunla bir bağ kuracak, ana düşmanı öldürmesi ile beraber de son bulacak bir hikaye olacak. Bunun yanında oynanışın da akıcı olmasını hedeflemekteyiz. Son zamanların popüler en oyunlarından biri olan “League of Legends” tarzında akıcı animasyon ve haraket sistemi çekirdek oyun yapısını destekleyecek ve keyiflı kılacak ana elementlerden olacaktır. Oynanış oyuncudan yüksek koordinasyon ,durumu okuma becerisi ve güçlü refleksler gerektirecek. Stratejik düşünme ve bu düşünceyi hızlı ve etkili bir biçimde uygulayarak düşmanların üstesinden gelme öne çıkan bir element olcak. Karakter ve efsanevi kılıcın güçlenmesinin yanı sıra oyunucunun da oyuna alışması ve daha iyi oynaması, sürekli daha zor düşmanlarla karşılaşması ve zafer kazanmasının da oyunumuzun doyurucu noktalarından biri olmasını planlamaktayız.

Oyunun yer alacağı zaman dilimi olarak içinde fantastik ögeler barındıracak orta çağı seçtik. Mekan olarak ise spesifik bir seçimimiz yok ancak ambiyans açısından 3-4 farklı tarzda mekan tasarlamayı ve bunları hikayeye yedirmeyi planlamaktayız. Örnek olarak ağaçlı ve dağlık bir orman, karlı bir dağ, plaj ve harap olmuş bir köy verilebilir.

1.1. Unity Collaborate

Oyun geliştirme projemizi 2 kişi yürütmekteyiz. İlk başladığımızda birbirimizin yaptığı işleri bilgisayarlarımızdaki bütün proje dosyalarını birbirimize ileterek yürütmekte idik. Bu süreç hem çok uzun sürmekte hem de tam doğru olarak çalışmamaktaydı. Unity versiyonlarımız farklı olduğundan uyumsuzluklarla karşılaşmaktaydık. Ayrıca proje yeniden derlenirken unity ayarlarımız aynı olmadığı için hatalar ortaya çıkmaktaydı.

Bu sorunun üstesinden gelmek adına araştırmalarımıza başladık. Karşımıza 2 seçenek çıktı. Bunlardan birincisi Unity'nin içinde hazır bulunan "Unity Collaborate"idi. İkincisi ise projeyi "Github" platmormuna yükleyerek Uniyden daha bağımsız ancak daha fazla kontrol sağlayan bir seçenekidi. Github platformunu kurması daha zor olduğundan ve projemizin bulunduğu aşamada buna ihtiyacımız olmadığından dolayı Unity içindeki sistemi tercih ettik. Unity Collaborate projeleri destekleyen, çalışanların kolayca koordine olmasını sağlayan, eş zamanlı olarak projenin farklı alanlarında çalışılmasına olanak sağlayan ,bağımsız geliştiricilere ve öğrencilere bazı limitlere kadar ücretsiz sunulan bir araç. Bu limitler ise 3 kişiyi aşmamak ve proje boyutunun 1 GB'tı aşmaması.

Bu sistemi kurarken ise şu aşamalardan geçtik. Projede çalışanlardan biri "owner" statüsünde diğer çalışanlara davet atmalı. Mail gelen çalışanlar kendi Unity hesaplarından bu davetleri kabul etmeli. Daha sonra owner daveti kabul eden çalışanlara bir proje rolu atamalı ve onlara bir koltuk ayırmalı. Proje rolu user,manager ve owner olabir. User çalışan ve manager ise menejerlik görevlerini üstlenen roldür. Takımımız 2 kişiden oluştuğu için Cudy owner ben ise manager rolünde çalışmalarımızı sürdürmekteyiz. Koltuk ataması ise Unity'nin bize sunduğu limitler ile alakalı. Ücretsiz Unity Collaborate versiyonunda 3 koltuk bulunmakta ancak bu koltukların alabileceği rollerde bir kısıtlama yok. 3. Çalışandan sonra bir çalışan daveti kabul edip geldikten sonra boş koltuk kalmadığı için sadece Guest rolunde gözlemci olarak durabilir ancak proje dosyalarını göremez ve ya değiştiremez. Bu kişinin de projeye dahil olabilmesi için hali hazırda projede çalışan 3 kişiden birinin ayrılması ve ya owner tarafından atılması gerekli.

Gereken kurulum aşamalarını bu şekilde tamamladıktan sonra Unity Collaborate'in işimizi nasıl kolaylaştırdığından bahsedelim. İlk kişi projeyi oluşturduktan sonra bunu

commit ederek bulut sistemine yüklüyor. Bu eylem diğer üyelerin sync ederek buluttan bu kopyayı indirmesine olanak sağlıyor. Bu işlem sadece projede yapılan değişiklikleri kapsadığı için büyük boyuttaki texture ve animasyon dosyaları tekrardan indirilmemiş oluyor. Sadece değişiklik yapılmış dosyalar taranıp bu dosyalar kullanıcılar arası exchange ediliyor. Proje içindeki objeler ve objeye ait component ayarları da tutulduğu için herhangi bir uyumsuzluk olmadan tek tuş ile çalışmalarımızı birbirimize aktarabiliyor ve eşzamanlı olarak farklı alanlarda çalışabiliyoruz.

2. OYUN GELİŞTİRME

2.1. Karakter

2.1.1. Karakter Seçimi:

Oyunda sadece bir karakter (warrior) bulunmaktadır. Bu karakter diğer düşmanlardan saldırılıyor, bu yüzden kendini savunmaya çalışması lazım ve bunu yapabilmek için bazı koruma silahları bulunması gerekir. Biz kılıç ve koruma kalkanı silahları gibi kullanmaya karar verdik. Bu karakteri seçmek için Unity Asset Store sitesinden arama yaparak ARPG oyununa en uygun 3D karakterini ve istediğimiz özellikler ile birlikte bulduk. Aşağıda resimi gösterilecektir.

Şekil 2.2.1.1. Oyun Karakteri

Bu karakter ile birlikte resimdeki mevcut olan silahları ve animasyonları bulunur. Biz bu karakteri import ederek Unity üzerine indirebildik ve modelini sahneye koyarak ekleyebildik. Bu model Player objesini adlandırdık ve Inspector kısmındaki bazı componentleri ekledik, onlara bahsedecek olursak; NavMeshAgent component, Bu bileşen, kullanarak sahnede gezinmesine izin vermek için oyundaki karaktere eklenir. Capsule Collider ve Rigidbody component'ler, Unity ortamında bir nesnenin başka bir nesneyle çarpıştığını algılamak için nesnelerin üzerine bu componentleri eklememiz lazımdır. Player-motor ve Player adlı bileşenleri, script olarak kodlama yapmak için ekledik. Bunları ekleyip gerekli adımlar yaparak istediğimiz hareketleri oluşturabiliriz. Oyunun karakteri seçtikten sonra onun hareketini nasıl ayarladığımızı bahsedeceğiz.

2.1.2. Karakter Hareketi:

Karakter, haritada herhengi bir yere mouse ile tıkladığımız zaman hareket etmeye başlıyor fakat haritada yürünebilir ve yürümeyebilir yerler var, O yüzden bu hareketi yapmak için iki tane script kullandık, birisi player-motor; bu scriptte NavMeshAgent tipinden bir değişken tanımladık, bu tip (type) Unity'de mevcut ve oun tanımlamak için UnityEngine.AI çağırdık çünkü oyunumuzda karakterin hareketi yürünecek yerlere gitmesi lazım o yüzden bu tip ve yapay zeka yardımıyla hangi yerleri yürümek için belli ediyor, Unity paltformunda Ground objesinde Navigation kısmı var, bu kısımdaki Bake özelliği Agentin yarıçapı ve yüksekliğini belirliyor, Object kısmı ise haritadaki tüm yerler yürünebilirliğinin veye yürünmeyebilirliğinin özelliğini gösteriyor.

Ayrıca bu scriptte iki tane fonksiyon var, Start fonksiyonu; ilk frame güncellemesinden önce çağrılan bir fonksiyondur. Bu metod içinde NavMeshAgent tipli olan değişkene GetComponent bileşeni atadık, bu bileşen alınacak component türünü döndürecek, bu durumda NavMeshAgent argüman olarak verdik. Tanımladığımız MoveToPoint fonksiyounu ise, Vector3 tipli bir nokta argüman olarak alıyor bu sayede x,y ve z eksenleri üzerinde bu noktanın değerler alması lazım. Bu fonksiyon içerisinde NavMeshAgent'in fonksiyonlarından birisi SetDestination kullandık ve bu metoda belirlediğimiz noktayı da argüman olarak atadık.

Diğer script ise player script'idir; burada birkaç değişkeni tanımladık,birisi camera tipinden bir değişken ve onu bu scriptin içinde kullanma sebebi ise karakterin herhengi bir yere hareket ettiği zaman camera onu takip edecektir. Camera ayarlarını daha sonra bahsedilecektir. Diğer değişkenin ise daha önce tanımladığımız player-motor script'i tipi olarak kullandık çünkü burada karakterin hareketi ve camera yönüyle birlikte mouse butonları kullanarak tıkladığımız yerlere gitmesi için yürünecek yerleri öğrenmek lazım.

Start fonksiyonunu da var burada ve onun içinde tanımladığımız camera değişkene Camera.main değişkenini atarak bizim ana kameramızı yaptık, bu main'i Kameranın Unity'deki essas özelliklerinden birisidir. Ayrıca tanımladığımız player-motor tipli değişkene GetComponent kullanarak istediğimiz bileşeni atadık.

Eğer mouse'un sol butona tıkladıysak ve tıkladığımız nesnenin üzerine player hareket edebilirse o zaman yerini değiştirilecek ve camera onu takip edecek.

Bunu yapmak için iki tane if statement'lerini kullandık, birinci olanı mouse'un sol butonu tıklandığı halde başlayacak, mouse'un sol butonuna tıklamak için Input.GetMouseButtonDown(0) metodunu kullandık, 0 sayı sol butonu demektir.

Bu if durumuna girdiyse, Ray tipinden bir değişken oluşturulacak ve bu ışını kameramızdan tıkladığımız yere doğru yönlendirmek isteriz demektir. Bu değişkene ScreenPointToRay adlı bir fonksiyon atadık, bu fonksiyon camera özelliklerinden birisi olur, ona argümen olarak mouse pozisyonu verdik çünkü işaretlemek istediğimiz noktayı o dur. Tıkladığımız nesne hakkında bazı bilgiler hafızada saklamak için RayHit tipinden bir nesne oluşturduk ve bundan sonra ikinci if statement'e girecek, bu durumu eğer işaretlediğimiz nesnenin yürünebilecek özelliğine sahip olduğunu kontrol etmek için kullanıldı, bunu gerçekleştirmek için Physics.Raycast metodun yardımıyla oldu çünkü bu adımda sakladığımız bilgileri ve tıkladığımız yerlerden yaralanabiliriz. Bu fonksiyon; mouse butonuyla tıkladığımız yer, yerin bilgileri, maksimum mesafe ve LayerMask tipinden bir değişken; olmak üzere 4 tane argüman aldı. Tıkladığımız yer ve onun bilgileri daha önce bahsettik, maksimum mesafe; karakterin yerinden mouse ile tıklandığı yerin arasındaki mesafedir ve biz 100 olarak tanımladık, LayerMask ise bu mask'ı Physics.Raycast kullanıldığı için koyduk ve onun içinde istediğimiz nesneleri atmamızı sağlayan bir tiptir. Bizim oyunumuzda istediğimiz nesneleri yürünebilecek yerler demektedir. Tabiki bu nesneyi atmak için Unity platformundan gerçekleştirilecektir. Karakter nesnemize gidip inspector kısmındaki bulunan palyer scriptinde tanımladığımız LayerMask değişkeninin yanında bir kutu var ve bu kutuya yürünebilecek yeri Ground adlı nesneyi attık. Bu layer'ler birbiriyle eşleştirmek için Ground nesnenin Layer kısmına Ground adlı bir layer de eklememiz lazım. Bunlar hepsini ikinci if statement içinde kontrol ediliyor ve eğer sonuç true ise o zaman tıkladığımız yeri point olarak tanımlayıp player-motor tipinden değişkeninin tanımladığımız MoveToPoint fonksiyonuna atarız ve böylece karakter hareket edebilir.

Hareket etmek için sadece sol butonu kullanık bu nedenle ayrı bir if statement tanımladık ve bu if'in içinde eğer sağ butonuna tıklama durumu olursa inceleme yapılacak çünkü sağ butonu etkileşim için olacak,o yüzden hiç bir şey olmayacak ve karakterin yerini değiştirilmeyecek.

Aynı adımlar yaptık burada, 2 tane if statement'lerini kullandık fakat bu durumda Input.GetMouseButtonDown(1) kullanıldı ve 1 sayı sağ butonu demektir.

Daha sonra kamera yerinden tıkladığımız yere doğru ışının gitmesi için Ray tipinden bir değişken oluşturduk ve aynı bir şekilde mouse pozisyonu argüman olarak ScreenPointToRay fonksiyonuna atadık ve RaycastHit değişkenine nesnenin bilgleri verdikten sonra ikinci if'e girdik ve burada sadece tıkladığımız yer, onun bilgileri ve maksimum mesafe Physics.Raycast fonksyonuna atadık çünkü bahsettiğimiz gibi bu kısım sadece etkileşim olarak yaptık ve hareket için ayarlamadık.

Böylece karakterin hareketini herhangi bir yere mouse ile tıkladığımız zaman, eğer bu yer yürünebilecek alanların kapsamında ise o zaman gerçekleştirilecek, kapsamında değil ise karakter aynı yerinde kalacak.

Hareket yöntemi bittikten sonra kamera ayarladığı ve nasıl yapıldığınıa dair bir açıklama yapacağız.

2.2. Kamera:

Kamera yönünün ayarlamasına ihtiyacımız var çünkü karakterin hareketini takip edecek, ona zoom yapacak ve etrafından dönebilme özelliğini olacaktır. Bunu yapabilmek için Unity'de Camera nesnenin içinde bir script component tanımladık, Camera Controller adlı bir scripttir. Onun içinde kodlama yaparak istediğimiz sonuca ulaşabildik. Bu adımlar tek tek bahsedilecektir.

Script içerisinde bazı değişkenler tanımladık ve onlar ya private yada public olarak yaptık çünkü Unity platformundan değiştirmek istediğimiz bazı değişkenler var, onların tipi public yaptık aksi takdirde aynı haline bırakmak istediğimiz ve Unity platformunda görünmesini istemediğimiz değişkenlerin tipi private yaptık.

Bu scriptte ilk olarak Transform tipinden bir değişken tanımladık bu değişken takip edilecek hedef olacak ve public olduğu için Unity üzerinde görünecek hale sebep oldu ve onun içine karakterimiz hedef olarak attık, diğer değişken ise Vector3 tipinden oluşturduk çünkü bu değişken kameranın hedeften uzaklığı olacaktır, Vector3 olma sebebi x,y ve z eksenlerinde bu uzaklığın ayarlanması lazım. Unity üzderinde bu ayarlama yaparak oyunumuza kameradan hedefe ve diğer yerlere en uygun uzaklığı bulduk, kameranın şuanki zoom ve dönme değerlerini taşıyan Float tipinde iki tane değişken tanımladık, bu değişkenler private olarak yaptık çünkü biz mevcut değerler ile değişklik yapmak istemiyoruz ve onları sadece güncelleme hesplamalarında kullanıldı çünkü oyun oynayınca kamera karaktere daha yakın bir

mesafeye yaklaştırmak ve karakter herhengi bir yere gidince kamera dönebilir bir şekilde olmak isteriz.

Kamera yönü karaktere yukarıdan nasıl bakacağı, zoom hızını ne kadar olacağı, maksimum ve minimum zoom değerleri ve dönme hızının değerini ayalayabilecek 5 tane Public Float tipiniden değişkenler tanımladık ve onlara sırayla 1, 4, 15, 5, ve 100 olmak üzere 5 tane değerler atadık, tabiki public olduğu için Unity platformundan bu değerler değiştirbiliriz. İki tane fonksiyon kullandık, onlar Update ve LateUpdate metodlarıdır. Update fonksiyonu her bir frame başına bir kez çağrılır, LateUpdate fonksiyonu ise Update metodu gibi ancak hemen onun ardından çağrılır.

Update fonksiyou içerisinde zoom ve dönme ayarlarını yaptık, karaktere zoom yapmak için fare kaydırma tekerliği ile olur o yüzden Input.GetAxis(“Mouse ScrollWheel”) kullanarak onun eksen değerini alıp daha önce belirlediğimiz zoom hızı değeri ile çarparak şuanki (current) zoom değerini güncellenmiş oldu ve bu hesaplama kısmında doğrudan dengelemek için (-=) işaretini kullanık. Daha sonra bu güncellenmiş zoom değeri, minimum ve maksimum zoom değerlerin arasında sıkıştırmak istiyoruz o yüzden Mathf.Clamp fonksiyonunu kullandık. Kamera karakter etrafından dönme değerini güncellemek için klavyedeki sol ve sağ okları yada A ve D harflere tıklayarak sol ve sağ taraflara dönebilir, bu dönme hareketi yatay bir şeklide olacağı için Input.GetAxis(“Horizontal”) onun eksen değerini alıp dönme hızı ve Time.deltaTime değerleriyle çarparak kamera dönme değerini elde ettik. Time.deltaTime son frame’den geçerli olana kadar saniye cinsinden bir aralık temsil ediyor bu nedenle hespalamada kullanıldı çünkü bu değere ihtiyaç duyduk. Zoom değerini güncellediğimiz gibi doğrudan dengelemek için (-=) işaretini kullandık.

LateUpdate fonksiyonu ise Update fonksiyondaki güncelleme hesaplamalardan sonra kameranın pozisyonu, yukarıdan bakma değeri ve karakter etrafından dönme değerini değiştirme adımları yaptık. Kamera pozisyonu için ilk olarak kamera ve karakterin arasındaki uzaklık değeri, güncellenmiş zoom değer ile çarparak onun sonucu hedef pozisyonundan çıkartarak elde edildi. Kamera yukarıdan bakmak için LookAt fonksiyonunu kullandık, bu fonksiyonda kamera karaktere yukarıdan mesafe uzaklığının değeri, Vector3.up değeri ile çarparak daha sonra hedef pozisyouna toplayarak istediğimiz sonuca ulaşabildik, Vector3.up kamera yukarı yönü demektedir. Son olarak kameranın karakterin etrafından dönmesi için RotateAround fonksiyonunu kullandık, bu metoda 3 tane argüman verdik; birinci argüman hedef

pozisyonu , ikinci ise Vector3.up çünkü yukarıdan da dönecek ve üçüncü olanı dönme açı değeri, burada bu değer şuanki (currant) dönme değerine eşittir. Bunu yaptıktan sonra elimizde karakterin haraketini takip eden bir kamera oldu.

2.3. Animasyonlar:

Karakter paketinde animasyonları bulunmaktadır ve bu animasyonları doğru bir şekilde kullanmak için gerekli adımları gerçekteştireceğiz çünkü bu karakterin animasyonlarının mevcut olmasına rağmen karakter sahne üzerinde koyunca ve Unity platformunda Play düğmesine tıklayına karakter yerinde duracak ve animasyonsuz gibi görünecek.

Bunu yapabilmek için Unity paltformunda Project kısmında Assets paketi var bu paketin içerisinde mouse sağ butonuna tıklayınca Animator Controller oluşturabiliriz ve onu oluşturduktan sonra platformda Animator kısmı görünecek ve oraya istediğimiz animasyonları koyabiliriz.

Karakterin içinde 4 tane animasyonu var, onlar idle, wake, charge ve attack animasyonlarıdır. Idle durumunda karakter yerinde kalacak, walk durumunda karakter yürüyecek, charge ise karakter daha hızlı bir şekilde yürümeye başlayacak ve son olarak attack ise karakter burada düşmanları saldırmaya çalışcak. Bu 4 tane animasyonları karakterindeki animations kısmında bulunur, onları kopyalayıp paketin dışarıya atıp daha sonra platformdaki Animator kısmına ekleyerek onları birbiriyle bağlamayı yaptık. Oyun başlayınca ilk olarak karakter yerinde olması lazım o yüzden temel animasyonu idle yaptık ve onunla wake animasyonu bağladık, bağlama yöntemi istediğimiz animasyonların arasında bir ilişki kurabiliriz ve onu yapmak için birinci animasyona tıklayıp ilişke kurma bir seçenek çıkacak, onu seçtikten sonra ok oluşturulacak ve bu oku diğer animasyona bağlamamız lazım. Oklar iki taraflı olabilir yani her iki animasyonu birbiriyle etkileyecektir. Wake animasyounla charge animasiyonu ve en son charge ile attack animasyonu bağladık. Onları bağladıktan sonra Animator kısmındaki Parameters kısmı var, bu kısımda; int, float, bool veya trigger 4 tane tipi olmak üzere bir değişkenler tanımlayabiliriz. isWalking, isCharging ve isAttacking, 3 tane bool tipi değişkenler tanımladık. isWalking değişkeni idle ve wake animasyonlarında arasındaki ilişkiyi için kullandık, aynı bir şekilde isCharging değişkeni walk ile charge arasında ve isAttacking charge ve attack arasında kullandık. Daha sonra idle'den çıkan oka tıklayarak sağ taraftaki inspector kısmındaki

conditions kısmı da bulunur, bu kısımda isWalking seçip onun değerini true yaptık, idle'e walk durumdan gelen oka ise isWalking değerini false dedik. Böylece walk ve charge arasındaki çıkan oka isCharging true ve gelen oka isCharging false olup charge'den attack'e çıkan oka isAttacking true ve attack'den charge'e giden oka ise isAttacking false yaptık.

Bu ayarlamaları yaptıktan sonra karakter objemize inspector kısmında Animator State Controller adlı bir script component oluşturduk. Bu script animasyonları kontrol etmek ve çalıştırmak için kullanıdı. Onun içinde bir tane Animator tipinden bir değişken tanımladık çünkü animator'a erişime ihtiyacımız olacak ve 3 tane int tipinden isWalking, isCharging ve isAttacking değişkenlerini tanımladık.

Start fonksiyonunun içerisinde GetComponent vasıtasıyla animator değişkenine Animator tipini atadık ve Unity paltformundaki Animator kısmındaki tanımladığımız 3 tane bool değişkenlerinden id'ler oluşturmak için Animator'un fonksiyonlarından birisini kullanmamızı gerekiyordu, bu metodun adı StringToHash, ona tanımladığımız bool değişkenleri string olarak argüman verdik, 3 tane değişken olduğu için 3 tane aynı tipten fonksiyon kullandık ve sonuçları yeni tanımlanan bir değişkenlere atadık ve böylece start metodunu bitirmiş olduk.

Update fonksiyonunu da çağırdık, bu metodun içerisinde animasyonları çalıştırmaya başladık, Animator tipli değişkeninin içerisinde tanımlanan bir fonksiyonunu da kullanık, boolean parametrenin değerini döndüren GetBool adlı bir fonksiyondur. Ona daha önce çıkarttığımız id'leri hash olarak değişkenler argüman verdik, bu nedenle 3 tane isWalking, isCharging ve isAttacking değerlerini elde ettik. 3 tane butonları tanımladık ve bu butonlara tıklayınca karakterin yürüme, hızlı yürüme ve saldırma animasyonları görünmeye başlayacak. Klavyedeki butonları oluşturmak için Input'un fonksiyonu GetKey(), 2 kere çağırdık, birisi yürümek için ve "w" argüman olarak aldı, diğerisi ise daha hızlı yürümek (charge) için ve o da "left shift" . Input'un fonksiyonu GetMouseButton(0) kullanıp mouse'un sol butonuna tıklayarak saldırma (attack) için ayarladık.

Bundan sonra animasyonları butonlar ile çalıştırma kodlamasını yazdık. Her bir animasyonu 2 tane if statement'i var. Bu durumları tek tek incelemek olurusak, yürümek için birinci if'in içinde eğer karakter yürmüyorsa yani isWalking boolean değişkeninin değeri false ise ve "w" butonu tıklanırsa karakter yürümeye başlayacak,

onu ayarlamak için animator'un SetBool metodunun kullanarak id hash değişkenine true değeri verdik. Aksi takdirde eğer karakter yürüyorsa ve "w" butonu bırakılırsa karakter duracak ve idle durumuna geçecek, burada id hash değişkeni false değerini aldı argüman olarak SetBool fonksiyonunda.

Karakter charge etmek için "w" ile birlikte "left shift" olması lazım, bunun için 2 tane if statement'lerini kullandık, birinci if' te eğer bu karakter daha hızlı yürümüyorsa yani isCharging boolean değişkeninin değeri false ve aynı zamanda "w" ve "left shift" butonlara tıklanırsa animator.SetBool çağırıp id hash değişkenine true değerini vererek karakter daha hızlı yürümeye başlayacak, ikinci if ise eğer hızlıca yürüyorsa ve "w" yada "left shift" butonlara tıklanmazsa karakter durması için SetBool fonksiyondaki argüman olarak is hash değişkenine false değerini verdik.

Son olarak karakter diğer düşmanları saldırmak için gereken butonlar "w", "left shift" ve mouse'un sol butonudur. Tekrardan 2 tane if statement'lerinde saldırıp saldırmadığını kontrol ediliyor. İlk if'in içerisinde eğer bu karakter saldırmıyorsa yani iaAttacking boolean değişkeninin değeri false ise ve onunla birlikte gerekli butonları tıklanırsa SetBool fonksiyonundaki id hash değişkenine true değerini atarak karakter hemen saldırmaya başlayacak. Fakat eğer bu karakter saldırıyorsa ve "w", "left shift" yada mouse'un sol butonlarını tetiklenmezse bu saldırmayı bitecek, o yüzden SetBool fonksiyonuna id hash değişkeni false değeri olarak bir argüman atadık.

Böylece karakterin 4 tane animasyonları, nasıl çalıştırdığını ve hangi butonlar kullanarak bu animasyonlar aktifleştirdiğini gösterdik.

2.4. Haritalar:

Oyunumuzun geçtiği mekanların ilgi çekici olması, sadece geçilip gidilen mekanlar olarak bakılmaması harita geliştirmeye başlarken bizim ana hedefimizdi. Bu bağlamda aklımıza gelen fikirlerden en beğendiklerimizi seçtik. Şu anda oyunumuzda 1 harita bulunmakta ve 3 tane daha eklemeyi hedeflemekteyiz. Yaptığımız ilk harita ise dağlık bir orman. Karakterimiz bu dağlık alanda patikalarda yürüyerek ulaşımını sağlayacak. Bu patikaların arasına keşif etmeyi seven oyuncular için gizli alanlar da ekledik. Eklemeyi düşündüğümüz alanlardan bazıları ise karlı dağlar ve deniz kenarı bir plaj. Oyunumuz orta çağda yer aldığı için herhangi bir kent ya da şehir koymayı

düşünmemekteyiz. Bunun yerine oyunun büyük bir kısmını açık alanda geçirerek doğal güzellikler ile göze hitap etmeyi planlıyoruz.

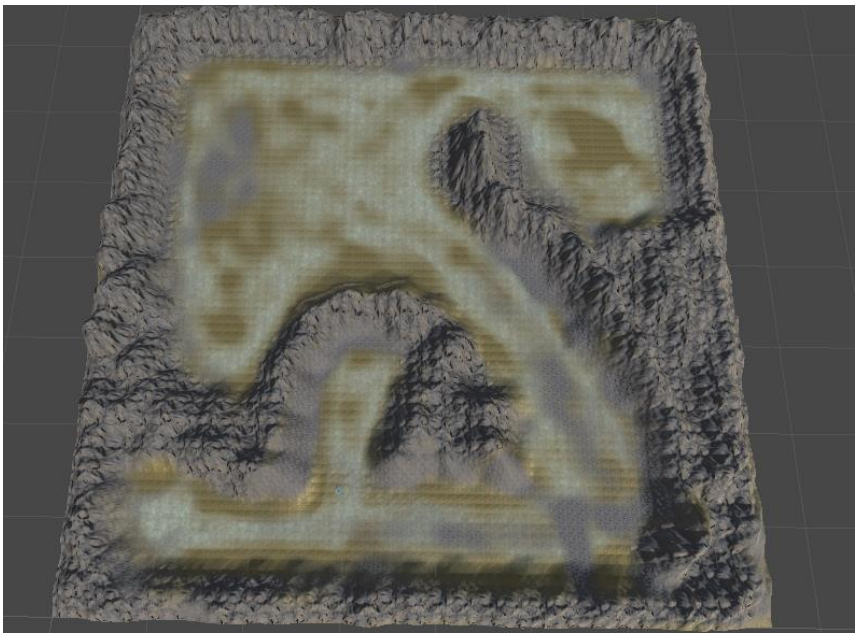
Haritamızı yapmaya araştırma ile başladık. Elimizdeki en iyi seçeneği ise yine unity içine dahil edilmiş ”Unity Terrain Editor” olarak belirledik. Hali hazırda projemiz için kullanabileceğimiz haritalar internette mevcuttu. Ancak bunlardaki problem haritaların ambiyansının uyuşmaması ve farklı dönemlere ve dünyalara ait olduğunun belli olması ve bunun oyunumuzda uzaklaştırıcı bir etki yaratacak olmasıydı. Bu haritaları kullanamayacağımız kararını verdikten sonra kendi haritalarımızı kolay ve efektif bir biçimde nasıl geliştirebileceğimiz ve çoğaltabileceğimizin yollarını aradık. Tam olarak da bu ihtiyaç için geliştirilmiş Unity Terrain Editor'u kullanmaya karar verdik.

Haritamızı oluşturmaya bir plain nesnesi oluşturarak başlıyoruz. Düz beyaz ve enine boyuna istediğimiz ölçüleri verebileceğimiz bir zemin olan bu nesneyi Unity'nin bize sunduğu araçlarla şekillendirerek, texture ekleyerek ve yürünebilir alanları belirleyerek oluşturuyoruz. Şekillendirmeyi bu paketin içinde bulunan fırcalar ile istediğimiz yere istediğimiz oranda uygulayarak yapıyoruz. Bu fırçaların büyüklüğü ve gücünü ne derecede kullanmak istediğimiz tamamen ayarlanabilir biçimde paketin içinde bulunmakta. Orneğin tepe yapıcı fırcayı alıp gücünü ve etki alanını kısarak basit bir tümsek yapabilirken, daha keskin bir yükselti fırcasını alıp alanını ve gücünü arttırarak bunu bir falez haline getirebilmekteyiz. Bu fırçaya “Raise or Lower Terrain” adı verilmekte. Buna ek olarak kolaylık olması için aynı fırçada “Set Height” modu da bulunmakta. Fırçamız ile çalışırken aynı yükseklikte zemin oluştırmak için çok dikkatli olunması gerektiği için bu özellik düşünülmüş. Yeryüzüne istediğimiz şekili verdikten sonra yürünebilir patikanın belli olması için seçtiğimiz alanı o anki değerinden bağımsız olarak seçtiğimiz yüksekliğe eşit bir düz alan haline getiriyor. Bu da hem oyuncunun yolunu bulmasını kolaylaştırıyor, hem de bizim harita üzerinde oyuncunun haraket edebileceği alanları belirlerken işimizi daha kolay kılıyor.

Fırçamızın ikinci modu ise “Paint Terrain”. Bu mod ise seçtiğimiz texture (doku)'nun semless (ayrımsız) bir biçimde kaplanmasını sağlıyor. Bunu yine bir fırça aracılığı ile seçtiğimiz büyüklükte ve güçte yapıyoruz. Ornek olarak çok büyük ve güçlü bir fırça ile tüm haritamızı toprak zemine kapladıktan sonra, alanı küçültülmüş ama yine güçlü bir fırça ile istediğimiz seyreklikte çim ekleyebiliriz. En son olarak da hem alanı hem de gücü düşük bir fırça ile detaylı kar dokusu ekleyip eriyen kar dokusu elde edebiliriz. Bu dokuyu eklerken “metallic” ve “smoothness” özellikleri ile oynayıp

istediğimiz görüntüyü elde edebiliriz. Seamless özelliği ise 1024x1024 pixel boyutunda (2'nin herhangi bir katı olabilir) bir fotoğrafı ifade etmektedir. Bu fotoğrafı semless yapan özelliği ise yan yana veya üst üste konulduğunda birbiri ile uyuşması ve iki fotoğraf arasındaki geçişin belli olmamasıdır. Bu bize çok küçük bir resim dosyası ile tekrar eden bir doku yapabilme olanağı sunmakta ve texture dosyası boyutunu önemli ölçüde küçültmektedir. Yine haritanın farklı yerlerinde bu texture özellikleri oynayarak farklı görünmelerini sağlayabilir, her bir texture resminin haritada ne kadar alan kaplayacağını seçebilir yani örüntü boyutunu değiştirebiliriz. Örnek olarak aynı çim dokusuna kapladığımız deniz seviyesinde bir alan ve dağın yamacına yakın iki alan düşünelim. Yüksek alanın "metallic" özelliğinini arttırarak yüksek mekanlardaki nemli dokuyu yakalayabiliriz. İkinci bir örnek olarak ise bir plaj alanındaki kum ve denizin kum ile bitiştiği ıslak kumu düşünelim. Islak kum görünümünü elde etmek için yine metalik ve smoothness değerini arttırabilir, kumun ışık yansıtıcılığı değerini arttır, ve parlaklığını düşürürsek bu bize daha koyu ancak yansıtıcılığı daha yüksek bir doku verir.

Yine "Paint texture" fırçasına benzeyen bir fırça olan "Paint trees" fırçası da aynı texture dosyası gibi önceden 3D olarak modellenmiş bir ağaç nesnesini istediğimiz sıklıkta ve büyüklükte haritaya kolayca eklememize olanak veriyor. Burada yine ağaç nesnesini değiştirerek, farklı ağaç nesneleri ekleyerek etkileyici bir orman görünümü elde etmemize olanak sağlamakta.



Şekil 2.4.1 Orman ve Dağlık harita

Özet olarak elimizdeki bu araçları kullanarak ilk önce yeryüzüne şekil veriyoruz. Daha sonra bu alanı farklı dokular ile kaplayarak canlandırıyoruz. Oyuncunun yürüyebileceği yerlerin yüksekliğini eşitledikten sonra geriye kalan işimiz ise oyunucun nerelerde yürüyüp yürüyemeyeceği. Bunun için Terrain Editor'un içinde bulunan NavMesh bileşenini kullanmaktayız. Bu bileşen onu eklediğimiz objenin NavMesh kurallarına tabi olmasını sağlıyor. Bunu oyuncuya ekliyoruz. Haritamızı da argüman olarak yani oyuncumuzun üstünde gezineceği alan olarak gönderdikten sonra bir takım argümanlar belirleyerek haritada yürünebilir alanları yapay zeka ile belirlemiş oluyoruz. Bu argümanlar oyuncunun hangi yükseklikler arasında yürüyebileceği, yürüyebileceği eğim gibi argümanlardır. Bu aşamadan sonra önümüze yürünebilir alanların highlight edildiği bir harita geliyor. Yine bu alanlarda detay olarak düzeltme yapma olanağına da sahibiz. Bu alanlar içerisinde oyuncumuz istediği alana gidebilmekte ancak bu alanın dışına çıkılmak istendiğinde çıkılmak istenilen noktaya en yakın haraket edebildiği noktada durmaktadır. Harita üzerinde yaptığımız yer şekili değişikliklerinden sonra yürünebilir alanların tekrar belirlenmesi için bu işlemleri tekrar yapmamız gerekmekte.

Eğer oyun içerisinde haraket eden ve üstünde yürünemeyecek bir nesne yürünebilecek bir alana girerse (toprak kayması ile patikanın yürünemez hale gelmesi), bu objeyi statik yerine dinamik olarak işaretlemeli, bir NavMesh componenti eklemeli ve "Mark as non-walkable" olarak işaretlemeliyiz. Bu oyunun runtime içerisinde herhangi bir nedenden dolayı gerçekleşirse oyuncunun yürümemesi gereken bir kayanın içinden geçmesi gibi hataların önüne geçmemizi sağlıyor. Bu stratejinin tam tersi de uygulanabilir olup runtime içerisinde yeni yürünebilir alanlar da yapmak mümkün ancak bu kısımı etkin olarak kullanmayı düşünmüyoruz.

3. SONUÇ VE YIL SONU HEDEFLERİ

3.1. Sonuç:

Bu dönem boyunca yaptığımız çalışmaları sizinle paylaştık. 2 dönemlik bir proje aldığımız için bu dönem odağımız daha çok kullandığımız programlar ve aletleri öğrenmek, birlikte çalışmak için düzenimizi iyice oturtmak ve oyun için temellerimizi atmaktı. Bu temeller üzerine 2. Dönem çalışmalarımızı devam ettirecek ve oyunumuzu tamamlayacağız.

3.2. Hedefler:

Gelecek dönem hedeflerimiz:

1)Karakter haraketinin ve animasyonunun oturtulması.

2)Düşman karakterlerin oyuna eklenmesi ve dövüş sistemi.

3)Harita tasarımının çeşitlendirilerek 3-4 haritaya çıkılması.

4)Karakter ve kılıcı için gelişme aşamaları oyuna entegre edilmesi.

4. KAYNAKÇA:

1. Unity Documentation : https://docs.unity3d.com/Manual/index.html
2. Unity Asset Store : https://api.unity.com/v1/oauth2/authorize?client_id=asset_store_v2&locale=en_US&redirect_uri=https%3A%2F%2Fassetstore.unity.com%2Fauth%2Fcallback%3Fredirect_to%3D%252F&response_type=code&state=53998af6-6a72-4fd2-afa4-2b96cb5544cf
3. Unity Collaboration : https://unity.com/unity/features/collaborate
4. Unity Dashboard : https://dashboard.unity3d.com/login
5. Introduction to Game Development , Terrain Generation and Navigation https://books.google.com.tr/books?id=saq9AAAAQBAJ&printsec=frontcover&dq=Beginning+3D+Game+Development+with+Unity+4:+All-in-one,+multi-platform+game&hl=en&sa=X&redir_esc=y#v=onepage&q=Beginning%203D%20Game%20Development%20with%20Unity%204%3A%20All-in-one%2C%20multi-platform%20game&f=false
6. Unity Components, Camera and Player Characters : https://books.google.com.tr/books?hl=en&lr=&id=WfAWzVW9IK0C&oi=fnd&pg=PT16&dq=Unity+Game+Development+Essentials&ots=AV6NAyOLAm&sig=Gd5snzveC-i_RPTYz-_tTy_ca_4&redir_esc=y#v=onepage&q&f=false
7. Unity 3D Game Development and Animations : https://books.google.com.tr/books?hl=en&lr=&id=vFMggPqVnhcC&oi=fnd&pg=PT9&dq=how+to+animate+characters+%C4%B1n+unity&ots=zbyhY-aJPv&sig=7XWPaIpqJD1WAC5P9jqQ6i_k72A&redir_esc=y#v=onepage&q=how%20to%20animate%20characters%20%C4%B1n%20unity&f=false
8. Designing And Animating Characters : https://tigerprints.clemson.edu/cgi/viewcontent.cgi?referer=https://scholar.google.com/&httpsredir=1&article=3386&context=all_theses
9. Setting Up Animations On The Player Character And RPG Projects : https://books.google.com.tr/books?hl=en&lr=&id=LzgzEAAAQBAJ&oi=fnd&pg=PT11&dq=Unity+in+Action:+Multiplatform+game+development+in+C%23&ots=EUJR97P6Qx&sig=6Dr77018P_3CYhLtxG6BVvmuG20&redir_esc=y#v=onepage&q=Unity%20in%20Action%3A%20Multiplatform%20game%20development%20in%20C%23&f=false