



**İSTANBUL ÜNİVERSİTESİ CERRAHPAŞA  
MÜHENDİSLİK FAKÜLTESİ  
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**BİTİRME PROJESİ 1**

**2D ROUGELIKE GAME WITH AI**

Hazırlayan : Sema ŞEN 1306160004

Danışman : Doç.Dr. Zeynep ORMAN

**HAZİRAN- 2020**

## ÖNSÖZ

Bu proje procedural generation teknikleri kullanılarak procedural levellerin yaratıldığı ve oyuncu olmayan karakterlerin davranışlarında yapay zeka kullanıldığı 2 boyutlu Roguelike tarzı bir oyunun dizaynı ve geliştirmesi üzerinedir.

Roguelike, Rogue(1980) isimli oyundan ilham alınan yapılan oyunlarla karakterize olmuş bir oyun türüdür. Rol yapma oyunlarının bir alt türü olup, procedural şekilde generalize edilmiş labirentler (dungeon) , sıra temelli (turn-based) oyun , kare temelli(tile-based) grafikler ve oyuncunun sürekli ölüşü gibi karakteristik özellikleri vardır.

Procedural generation verilerin manuel yerine algoritmalar kullanarak yaratıldığı bir metottur.Bu metotta genellikle insan tarafından yaratılmış assetler ve algoritmalar kullanılarak bilgisayar tarafından rastgele bir şekilde yeni veriler üretilir.

Projenin amacı level dizaynında Procedural Generation ve oyuncu olmayan karakterlerin davranışlarında yapay zeka kullanılarak 2 boyutlu Roguelike tarzı bir oyunun Unity oyun motoru ile geliştirmesidir. Ayrıca proje bitiminde yapılacak olan sunumla roguelike tarzı oyun türünün ve oyunlarda yapay zeka kullanımının tanıtılmasıdır.

Günümüzde oyunlarda yapay zekanın kullanımı gittikçe artmaktadır.Bu proje ile roguelike tarzı bir oyuna yapay zeka ekleyerek oyunlarda yapay zekanın kullanımına katkıda bulunması ve proje bitiminde yapılacak olan sunumla roguelike tarzı oyun türünün ve oyunlarda yapay zeka kullanımının tanıtılması beklenmektedir.

Bu çalışma boyunca gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Sayın Doç.Dr.Zeynep ORMAN'a en içten dileklerle teşekkür ediyorum.

Sema Şen 1306160004

# İÇİNDEKİLER

ÖNSÖZ.....	I
İÇİNDEKİLER.....	II
ŞEKİL LİSTESİ.....	III
ÖZET.....	V
SUMMARY.....	VI
1. GİRİŞ.....	2
2. GENEL KISIMLAR.....	3
2.1. DIJİTAL OYUNLARIN GELİŞİMİ.....	3
2.2. ROGUELIKE TÜRÜ.....	3
2.3. PROCEDURAL LEVEL DESIGN.....	4
2.3.1. Spelunky Oyununun Procedural Level Dizaynında Kullanılan Yöntem.....	4
2.4. OYUNLARDA YAPAY ZEKA KULLANIMI.....	5
2.4.1. Hareket (Movement).....	5
2.4.2. Karar Verme ( Decision Making ).....	5
2.4.3. Strateji.....	5
3. KULLANILAN ARAÇ VE YÖNTEM.....	6
3.1. OYUNUN SENARYOSU VE TEMEL TASARIMI.....	6
3.2. ASSETLERİN BULUNMASI VE/VEYA TASARLANMASI.....	7
3.3. LEVEL DESIGN.....	10
3.4. OYUNCUNUN HAREKET VE KONTROLÜNÜN OLUŞTURULMASI.....	25
3.5. DÜŞMAN KARAKTERLERİN YARATILMASI.....	33
3.5.1. Burning Ghoul.....	34
3.5.2. Wizard.....	35
3.5.3. Demon.....	38
3.6. KULLANICI ARAYÜZÜ VE LEVELLERİN OLUŞTURULMASI.....	48
4. TARTIŞMA VE SONUÇ.....	58
5. KAYNAKLAR.....	59
6. ÖZGEÇMİŞ.....	61

## ŞEKİL LİSTESİ

- Şekil.3.1.1.Yeraltı Harabesi
- Şekil.3.1.2.Karakterler
- Şekil.3.1.3.Assetler
- Şekil.3.2.1. GothicVania Church Paketi [6]
- Şekil.3.2.2. Grotto Escape 2 paketi [7]
- Şekil.3.1.3. 2D Pixel Item Paketi [8]
- Şekil.3.1.4. Oyundaki Kalpler
- Şekil.3.2.5. Tiles
- Şekil.3.2.6.Odaların Yaratımı
- Şekil.3.3.1.Odacıklar
- Şekil.3.3.2.Odacık 1
- Şekil.3.3.3.Odacık 2
- Şekil.3.3.4.Odacık 3
- Şekil.3.3.5.Odacık 4
- Şekil.3.3.6.Odacık 5
- Şekil.3.3.7.LavaRoom Prefabı
- Şekil.3.3.8.LavaController kodu
- Şekil.3.3.9.LavaBall kodu
- Şekil.3.3.10.Spikes Roomdaki dikenler
- Şekil.3.3.11.LavaRoom daki lava
- Şekil.3.3.12.PlayerHealth kodu lava ve dikenlere çarpma durumu
- Şekil.3.3.14.KeySpawner kodu
- Şekil.3.3.15. Kapı ve Anahtar
- Şekil.3.3.16.Key kodu
- Şekil.3.3.17. Door Kodu
- Şekil.3.3.18.Oda Pozisyonları
- Şekil.3.3.19. Çözüm Yolu Ve Çözüm Yolu Dışında Kalan Odalar
- Şekil.3.3.20.LevelGeneration kodu
- Şekil.3.3.21.LevelGeneration kodu devamı
- Şekil.3.3.23.RoomType kodu
- Şekil.3.3.24.SpawnRoom kodu
- Şekil.3.3.25.Level örneği 1
- Şekil.3.3.26.Level örneği 2
- Şekil.3.3.27.Level örneği 3
- Şekil.3.3.27.Level örneği 4
- Şekil.3.4.1.Oyuncu Animasyonları
- Şekil.3.4.2.Oyuncu Animator Stateleri
- Şekil.3.4.3.Oyuncu Animasyon parametreleri
- Şekil.3.4.4.Player Controller kodu
- Şekil.3.4.5.Player Controller kodu devamı
- Şekil.3.4.6.Player Controller kodu devamı
- Şekil.3.4.7.Player Attack kodu
- Şekil.3.4.8.Player Attack kodu devamı
- Şekil.3.4.9.Player AttackPoint
- Şekil.3.4.10. Player Health kodu
- Şekil.3.4.11. Player Health kodu devamı
- Şekil.3.4.12. PlayerHealth koduna kalp resimleri eklenmiştir

Şekil.3.4.13.Oyuncunun Canını Gösteren UI  
Şekil.3.4.14.Oyuncunun Canını Gösteren UI , Oyun görünümü  
Şekil.3.5.1.Enemy Kodu  
Şekil.3.5.1.1.BurningGhoul  
Şekil.3.5.1.2.BurningGhoul kodu  
Şekil.3.5.1.2.BurningGhoul kodu devamı  
Şekil.3.5.2.1.Wizard  
Şekil.3.5.2.2.Wizard kodu  
Şekil.3.5.2.3.Wizard kodu devamı  
Şekil.3.5.2.4.Wizard FirePoint  
Şekil.3.5.2.5. FireBall  
Şekil.3.5.2.5. FireBall kodu  
Şekil.3.5.3.1.Demon  
Şekil.3.5.3.2.Demon Karakterinin Canının Ekranda Gösterilmesi  
Şekil.3.5.3.3.DemonHealth kodu  
Şekil.3.5.3.4. A\* Path finding  
Şekil.3.6.1.Oyundaki sahneler  
Şekil.3.6.2.GameManeger objesi  
Şekil.3.6.3.LevelComplete objesi  
Şekil.3.6.4.LevelComplete kodu  
Şekil.3.6.5.Menu sahnesi  
Şekil.3.6.5.Menu sahnesi bileşenleri  
Şekil.3.6.6.Button On Click() metottu  
Şekil.3.6.7.Menu kodu  
Şekil.3.6.8.Rules Sahnesi  
Şekil.3.6.9. Birinci , İkinci ve Üçüncü Sahneler  
Şekil.3.6.10. Örnek 1  
Şekil.3.6.11. Örnek 2  
Şekil.3.6.12. Camera View  
Şekil.3.6.13. Oyunun Oyun Modunda Görünüşü  
Şekil.3.6.14.Door kodu  
Şekil.3.6.15.Level4 oyun görünümü  
Şekil.3.6.16.Demon Hareket Haritası  
Şekil.3.6.17.Level4 örnek  
Şekil.3.6.18.Credits  
Şekil.3.6.19.Credits Button Onclick fonksiyonu  
Şekil.3.6.20.Credits Kodu

## ÖZET

Proje procedural generation teknikleri kullanılarak rastgele levellerin yaratıldığı ve oyuncu olmayan karakterlerin davranışlarında yapay zeka kullanıldığı 2 boyutlu Roguelike tarzı bir oyunun dizaynı ve geliştirmesi üzerinedir. Oyunun geliştirilmesinde Unity oyun motoru kullanılmıştır.

Roguelike, Rogue(1980) isimli oyundan ilham alınarak yapılan oyunlarla karakterize olmuş bir oyun türüdür. Rol yapma oyunlarının bir alt türü olup, procedural şekilde generalize edilmiş labirentler (dungeon) , sıra temelli (turn-based) oyun , kare temelli(tile-based) grafikler ve oyuncunun sürekli ölüşü gibi karakteristik özellikleri vardır.

Procedural generation verilerin manuel yerine algoritmalar kullanılarak yaratıldığı bir metottur. Bu metotta genellikle insanlar tarafından yaratılmış assetler ve algoritmalar kullanılarak bilgisayar tarafından rastgele bir şekilde yeni veriler üretilir.

Bu oyunda procedural generation kullanılarak, oyunun geçtiği dünyanın büyük bir kısmı bilgisayar tarafından her yeni oyunda yeniden yaratılır. Oyuncu oyuna her yeniden başladığında rastlantısal, daha önceki levellerden farklı, yeni bir levelle karşılaştığı için procedural generation kullanmak oyunun oynanabilirliğini arttırır ve oyuncunun daha fazla hata yapmasını sağlar.

Oyunun bir leveli  $4*3$ , 12 odacıktan oluşur. Bu odacıklar her oyuna yeniden başladığında belirli kurallar dahilinde rastgele bir şekilde dizilir. Böylece her seferinde benzersiz bir level tasarlanmış olur.

Oyunlarda kullanılan yapay zeka 3 gruba sınıflandırılabilir: hareket (movement), karar verme (desition making), ve strateji. Ama her oyun bu üç gruba sahip olmak zorunda değildir. Oyun türlerine göre sahip oldukları yapay zekalar değişir. Bu oyunda strateji grubu kullanılmamıştır.

Oyunda düşmana implement edilecek yapay zekada hareket grubunda düşman karakterin oyuncuyu takip etmesi için Unity'nin desteklediği a\* algoritması projesi kullanılarak düşmanın yapay zekasına Path Finding eklenmiştir.

Karar verme grubu için kullanılan en popüler teknikler; decision trees, state machines ve behavior trees'dir. Düşman karakterinin saldırı ve oyuncuyu bulma arasında karar vermesi için, oyuncu belirli mesafeye girdiğinde oyuncuya saldırması için, ve her zaman yaptığı harekete uygun animasyonların oynayabilmesi için Unity'nin içinde bulunan Animator sayfasındaki state machines kullanılmıştır.

Bunlara ek olarak düşman karakterinin sergilediği diğer tüm davranışları, her zaman oyuncunun bulunduğu yöne bakmak, rastgele süre aralıklarıyla atış yapmak , platformlar üzerinde ileri geri ilerlemek gibi, kodlar ile yazılmıştır.

Level dizaynında procedural generation ve düşmanlarda yapay zeka kullanılması, oyuna her yeniden başladığında farklı bir oyunla karşılaşmamızı sağlar buda oyunun tahmin edilebilirliğini azaltır ve oyunu oynamayı daha eğlenceli hale getirir.

## SUMMARY

This project consists of the development of a 2D Roguelike game in which procedural generation techniques are applied to create procedural levels, and artificial intelligence used to create more intelligent non-player characters by using the cross-platform game engine Unity.

Roguelike is a game genre characterized by games that were inspired from the game Rogue (1980). It is a subgenre of role-playing video game characterized by a dungeon crawl through procedurally generated levels, turn-based gameplay, tile-based graphics, and permanent death of the player character.

Procedural generation is a method of creating data algorithmically as opposed to manually, typically through a combination of human-generated assets and algorithms coupled with computer-generated randomness and processing power.

By using procedural generation in this game, most of the game world would be generated by the game for every new gameplay session. This helps to encourage replayability and complements permanent failure.

A level in this game is made of 12 rooms in a 4\*3 grid. For every new gameplay session a new unique level will be generated based on some rules.

Artificial intelligence in games can be group in to 3 classes which are movement, decision making and strategy. But not every game has to have all the of this AI groups. For every game genre use of AI will change. In this game strategy group will not be used.

In this game for enemy AI movement , for Enemy to find and follow player Path finding is used by implementing A\* algorithm project that Unity supports.

For decision making most popular techniques are decision trees, state machines and behavior trees. In this game for enemies to make a decision between attacking and finding the player and attacking when the player in certain distance and also for every movement that enemy makes for the playing the right animation, state machines inside the Unity Animator window is used.

Additionally all the other enemy behaviors , like always looking where the player is , attacking in random time intervals , patrolling on top of platforms are written by code.

By using procedural generation and artificial intelligence for every gameplay session a unique level will be generated . This makes the game harder to predict , more enjoyable to play and more interesting for the player. This increases the replayability.

# 1.GİRİŞ

Teknolojinin hızla gelişmesi, dijital oyun sektörünün büyümesini olumlu yönde etkilemiştir.1950lerde başlayan dijital oyun üretimi günümüzde büyük bir sektör haline gelmiştir.Bu sektörde büyük şirketler olduğu gibi kendi oyunlarını geliştiren bağımsız (indie) oyun geliştiricileride vardır.

Oyun geliştirilirken bir oyun motoruna ihtiyaç vardır. Bazı geliştiriciler bu motoru kendileri yazarken bazıları hazır oyun motorlarını kullanır. Günümüzde birçok büyük çaplı hazır oyun motoru vardır.Bunlardan en çok bilinenleri Unity ve Unreal Engine dir. Bu projede oyunun geliştirmesinde Unity oyun motoru kullanılmıştır.

Bu projede 2 boyutlu roguelike tarzı bir oyun geliştirilmiştir. Roguelike, Rogue(1980) isimli oyundan ilham alınarak yapılan oyunlarla karakterize olmuş bir oyun türüdür. RPG oyunlarının bir alt türü olup , procedural şekilde generalize edilmiş labirentler (dungeon) , sıra temelli (turn-based) oyun , kare temelli(tile-based) grafikler ve oyuncunun sürekli ölüşü gibi karakteristik özellikleri vardır.

Oyunda randomize leveller oluşturulması için procedural generation tekniği kullanılmıştır. Procedural generation verilerin manuel yerine algoritmalar kullanarak yaratıldığı bir metottur.Bu metotta genellikle insan tarafından yaratılmış assetler ve algoritmalar kullanılarak bilgisayar tarafından rastgele bir şekilde yeni veriler üretilir. Bu oyunda procedural generation kullanarak , oyunun geçtiği dünyanın büyük bir kısmı bilgisayar tarafından her yeni oyunda yeniden yaratılır. Oyuncu oyuna her yeniden başladığında raslantısal , daha önceki levellerden farklı , yeni bir levelle karşılaşır.

Günümüzde oyunlarda yapay zekanın kullanımı gittikçe artmaktadır. Bu oyunda,düşmanların hareketleri ve karar vermeleri için yapay zeka kullanılarak düşmanlara akıllı davranışlar kazandırılmıştır.



## **2.GENEL KISIMLAR**

### **2.1.DİJİTAL OYUNLARIN GELİŞİMİ**

Video oyunlarının gelişimi ilk olarak 1960'larda Amerika'da başlamıştır. Atari Amerika'da kurulan ilk oyun şirketidir. Amerika'dan sonra oyun pazarına Japonya Nintendo ve Sony şirketleriyle girmiştir. 1980'lerde bulunan oyun konsolları pazarın büyümesine neden olurken 2000'li yıllarda kişisel bilgisayarların yaygınlaşması ile oyunların CDlerde depolanmasıyla bilgisayar oyunları popülerliği artmıştır. Günümüzde bilgisayar oyunları , Xbox 360, PS3 ve Wii konsollardaki oyunlar ve mobil oyunlar giderek oyun pazarını büyültmektedir. [1]

### **2.2.ROGUELIKE TÜRÜ**

Roguelike, Rogue(1980) isimli oyundan ilham alınan yapılan oyunlarla karakterize olmuş bir oyun türüdür. Rol yapma oyunlarının bir alt türü olup , procedural şekilde generalize edilmiş labirentler (dungeon) , sıra temelli (turn-based) oyun , kare temelli(tile-based) grafikler ve oyuncunun sürekli ölüşü gibi karakteristik özellikleri vardır.

Roguelike oyunlar, rastgele yaratılmış tile bazlı haritalara, yaratıklara ve eşyalara sahiptirler ve bunlar her yeni oyunda en baştan yaratılır. Yaratılan haritalar ise tipik olarak koridorlarla birbirine bağlanan odaları içerir. Bunlardan bazıları hazine odaları veya yaratık inleri gibi önceden ayarlanmış odalar olabilirler.

Roguelike'lar geleneksel olarak “permadeath” oyunlardır. Yani oynadığınız karakter öldüğünde, oyun biter. Yaratılan yeni karakter de yeni bir evrende yaratılacaktır.

Rogue (1980) oyunu roguelike türünün öncüsüdür.

Spelunky (2008) oyunu procedural generation kullanıldığı en güzel roguelike örneklerinden biridir. Bu oyunda Spelunky' nin procedural generation modeli kullanılarak 4\*4 grid de 16 odacıktan oluşan bir level dizaynı oluşturulacaktır.

Rogue Legacy(2013) oyunu , oyuncu oyunu her kaybedip levele tekrar başladığında sadece level dizaynını değiştirmekle kalmayıp aynı zamanda oyuncuyu da bir kraliyet aile ağacından değişik aile üyeleriyle değiştiren türünün yenilikçi örneklerinden biridir.

Diğer popüler roguelike oyunları The Binding of Isaac(2011) , Dead Cells(2017) , Crypt of the NecroDancer (2015) bu türün önde gelen diğer örnekleridir. Ancak bu türde onlarca başka oyunlarda bulunmaktadır ve hergün yenileri eklenmektedir. [2]

## **2.3.PROCEDURAL LEVEL DESIGN**

Procedural generation verilerin manuel yerine algoritmalar kullanılarak yaratıldığı bir metottur. Bu metotta genellikle insan tarafından yaratılmış assetler ve algoritmalar kullanılarak bilgisayar tarafından rastgele bir şekilde yeni veriler üretilir.

Procedural generation level dizaynı için kullanıldığında, oyunun geçtiği dünyanın büyük bir kısmı bilgisayar tarafından her yeni oyunda yeniden yaratılır. Oyuncu oyuna her yeniden başladığında raslantısal , daha önceki levellerden farklı , yeni bir levelle karşılaşır.

Procedural generation tekniği ilk olarak rogue(1980) oyunu ile birlikte oyunlarda dizayn için kullanılmaya başlanmıştır. Zamanla diğer oyun türlerindedeki kullanımı artmaktadır. Oyunlar için eşsiz content üretimi ( karakter dizaynı , level dizaynı , oyuncunun kullanabileceği itemler gibi ) büyük bir zaman ve emek gerektiren bir iştir. Procedural generation tekniği level dizaynında kullanıldığında ise bu yükün büyük bir kısmı ortadan kalkar. Çünkü eşsiz level dizaynını bilgisayar yapar. [3]

### **2.3.1. Spelunky Oyununun Procedural Level Dizaynında Kullanılan Yöntem**

Bu oyunda procedural generation kullanarak , oyunun geçtiği dünyanın büyük bir kısmı bilgisayar tarafından her yeni oyunda yeniden yaratılır. Oyuncu oyuna her yeniden başladığında raslantısal , daha önceki levellerden farklı , yeni bir levelle karşılaşır.Procedural design şu şekilde gerçekleştirilmiştir:

Oyunun bir leveli 4\*4 grid’de, 16 odacıktan oluşturulmuştur. Level dizaynında ilk önce çözüm yolu (solution path ) oluşturulmuş daha sonra geri kalan boş gridlere raslantısal odacıklar atanmıştır. Odacıklar 4 tipten birindendir.

0:kenar odacıklar , bu odacıklar çözüm yolu üzerinde olmayacaklardır.

1: RL odacıklar right ve left kenarlarında çıkış olan odacıklardır

2: BRL odacıklar bottom , right ve left kenarlarında çıkış olan odacıklardır

3: TRL odacıklar top , right ve left kenarlarında çıkış olan odacıklardır

Çözüm yolu bu yol üzerindeki odacıkların birbirine değen kenarlarında çıkış olmasıyla sağlanır.Oyuna her yeniden başlandığında çözüm yolu ve bu yol üzerindeki odacıklar yeniden yaratılır. [4]

## **2.4.OYUNLARDA YAPAY ZEKA KULLANIMI**

Packman yapay zekanın kullanıldığı ilk oyundur. İlk defa bir oyunda düşman karakterler oyuncu gibi hareket ediyordu ve oyuncunun hayatını zorlaştırmaya çalışıyordu. Packmannin kullandığı yapay zeka yöntemi state machines di. 4 tane düşan karakterinin değişik özellikleri vardı ve bazıları oyuncuyu kovalarken bazıları kaçıyordu.

İlerleyen zamanla birlikte oyunlarda kullanılan yapay zeka yöntemleride arttı. Günümüzde oyunlarda kullanılan yapay zeka 3 gruba sınıflandırılabilir : hareket (movement) , karar verme (desition making) , ve strateji. Ama her oyun bu üç gruba sahip olmak zorunda değildir. Oyun türlerine göre sahip oldukları yapay zekalar değişir.

### **2.4.1.Hareket (Movement)**

Hareket sınıfı , kararları hareketlere çeviren algoritmaları içerir. Eğer bir düşman karakteri oyuncuya saldırmak istiyorsa önce ona yaklaşmalıdır veya ateş ediyorsa oyuncuya dönük olmalıdır. Oyuncuyu takip etmesi , oyuncudan kaçması , içinde bulunduğu dünya ile iletişim halinde olması , mesela önünde bir engel varsa durup geri dönmesi hareket grubuna giren yapay zeka örnekleridir.

Hareket grubunda çoğu zaman hareketlerin gerçekleştirilmesi animasyolarla sağlanır. Animasyonun başlangıcı için koddan komut verilir ve animasyon içerisindeyken başka komutlar için triggerlar konulabilir. Mesela saldırı animasyonun 3 saniyesinde oyuncuya zarar verme komutuna trigger yollanabilir. Böylece animasyonun tam oalrak doğru yerinde oyuncu zarar görmüş olur.

### **2.4.2. Karar Verme ( Decision Making )**

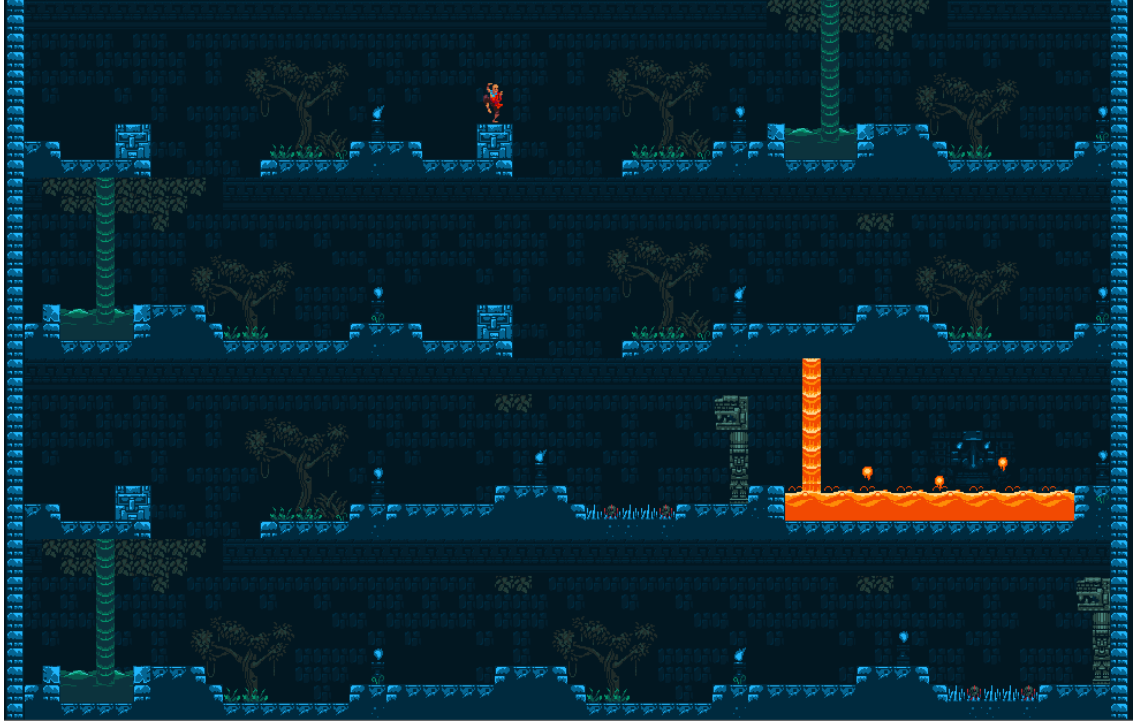
Karar verme , karakterin bir sonraki adımda ne yapacağına karar vermesidir. Örnek vermek gerekirse oyuncuya belirli yakınlıktayken saldırması , canının azaldığı zaman oyuncudan kaçması , oyuncuyu aramak için odacıkları gezmesi verilebilir. Karar verme grubu için kullanılan en popüler teknikler; decision trees, state machines ve behavior trees'dir.

### **3.4.3. Strateji**

Çoğu oyun strateji bölümüne ihtiyaç duymaz. Düşmanlara vermek istedikleri davranışları hareket ve karar verme ile verebilirler. Ama bir grup karakteri kordine etmemiz gerektiğinde strajeji grubu kullanılması gerekir. Burda karakterlerin kendi hareket ve karar verme davranışları vardır ama bunları grubun stratejisine uyucak sekilde yaparlar. [5]

### 3.KULLANILAN ARAÇ VE YÖNTEM

#### 3.1. OYUNUN SENARYOSU VE TEMEL TASARIMI

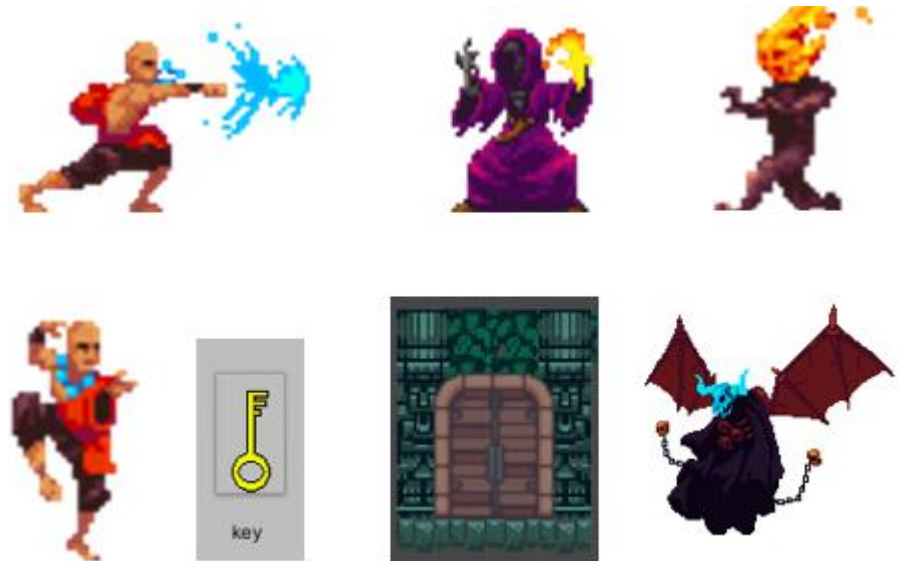


Şekil.3.1.1.Yeraltı Harabesi

#### ESCAPE FROM THE RUINS

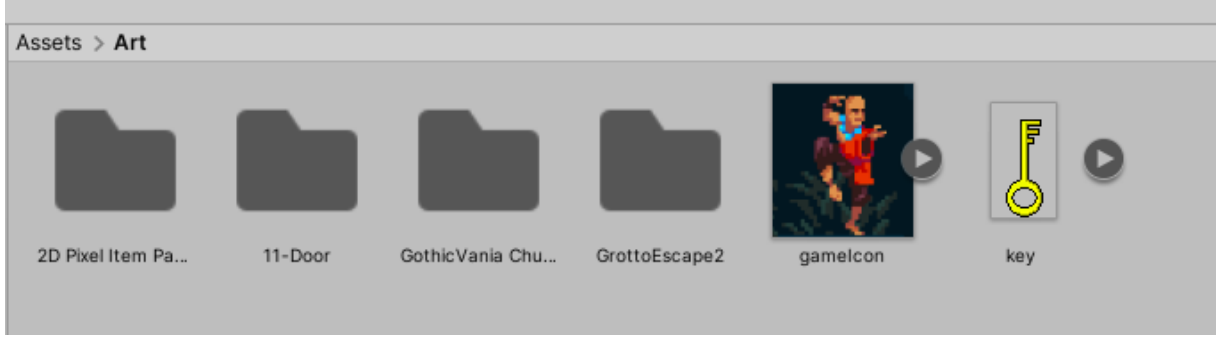
Spelunky (2008) adlı oyundan esinlenilmiş olan bu oyunda amaç Şekil.3.1.1'deki yeraltı harabesinden kaçıştır.

Ana karakter (oyuncu) savaşçı bir keşiftir. Oyuncu oyuna başladığında kendini rastgele yaratılmış bir labirentin içinde bulur. Oyuncunun amacı anahtarı bulup kapıya ulaşmaktır. Kapıya ulaştığı zaman eğer anahtarı bulmuşsa kapı açılır ve bölüm tamamlanır. Şekil.3.1.2.'de karakterler gözükmemektedir.



Şekil.3.1.2.Karakterler

### 3.2. ASSETLERIN BULUNMASI VE/VEYA TASARLANMASI

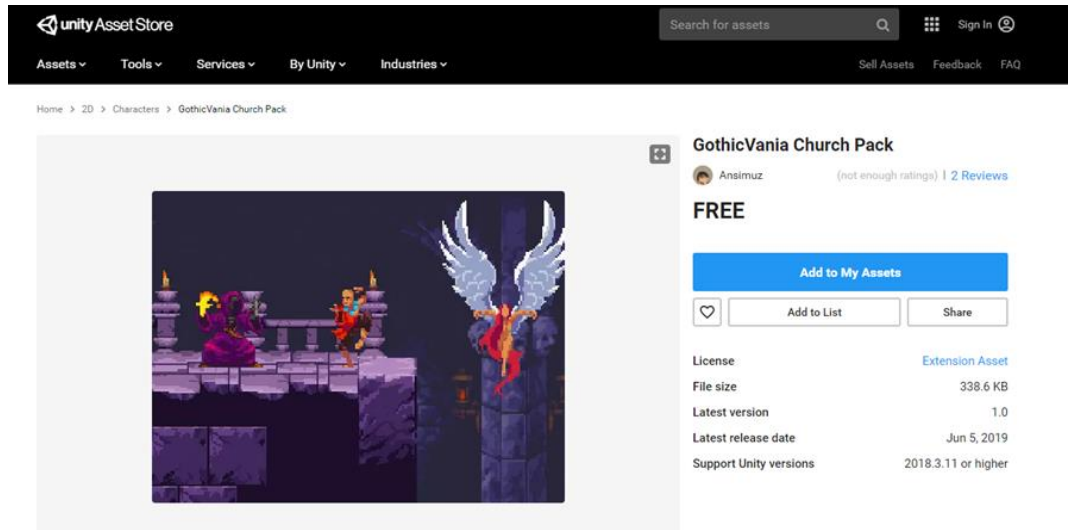


Şekil.3.1.3.Assetler

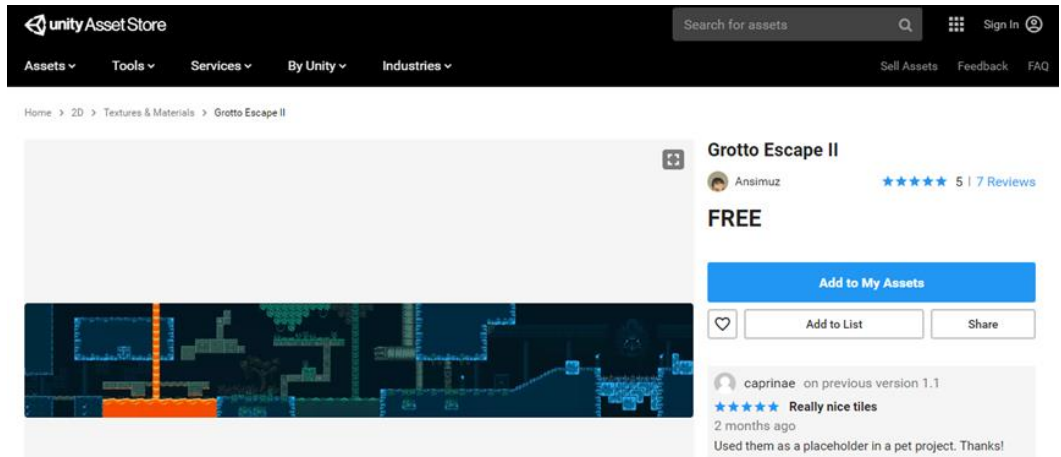
Bu oyunda kullanılan assetler ( Şekil.3.2.1 ) Unity Asset Store’den bulunmuştur.

Kullanılan her paketler ücretsiz olup oyun için gereken assetleri karşılamaktadır.

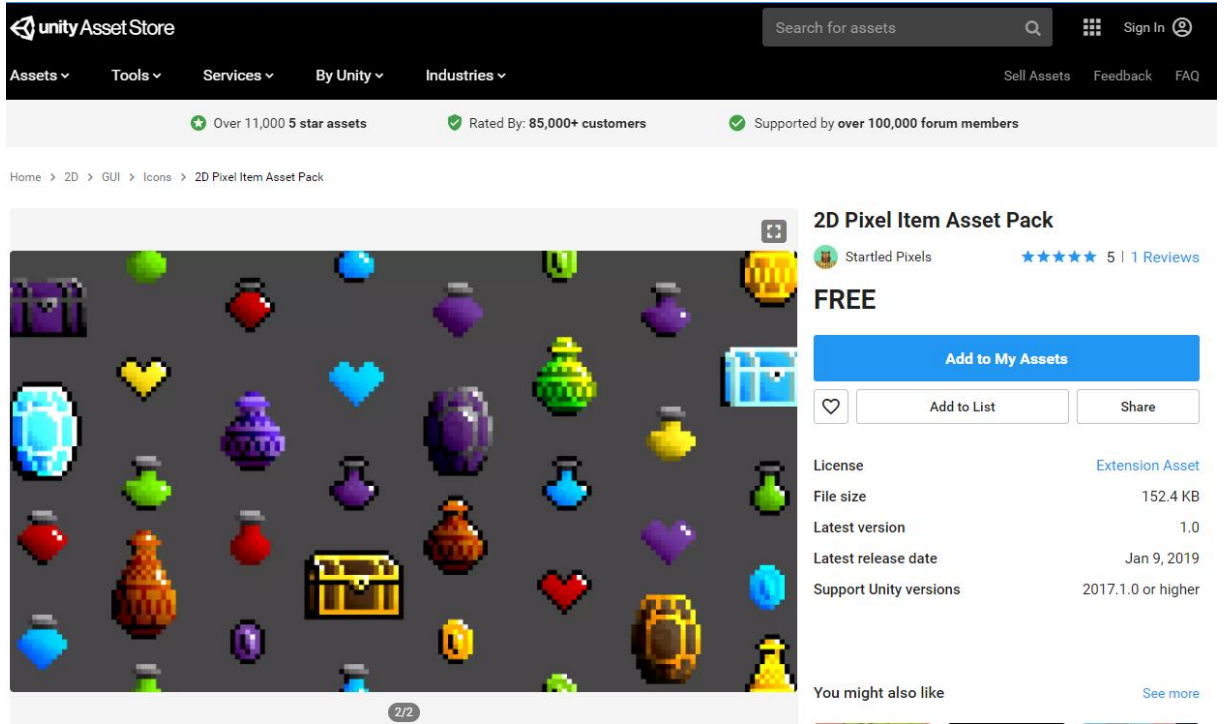
GothicVania Church paketinden (Şekil.3.2.2) oyuncu ve düşman karakterleri kullanılırken, Grotto Escape 2 paketi (Şekil.3.2.3) level dizaynı için kullanılmıştır. 2D Pixel Item paketinden (Şekil.3.2.4) kalp , key ve kapı kullanılmıştır.



Şekil.3.2.1. GothicVania Church Paketi [6]



Şekil.3.2.2. Grotto Escape 2 paketi [7]

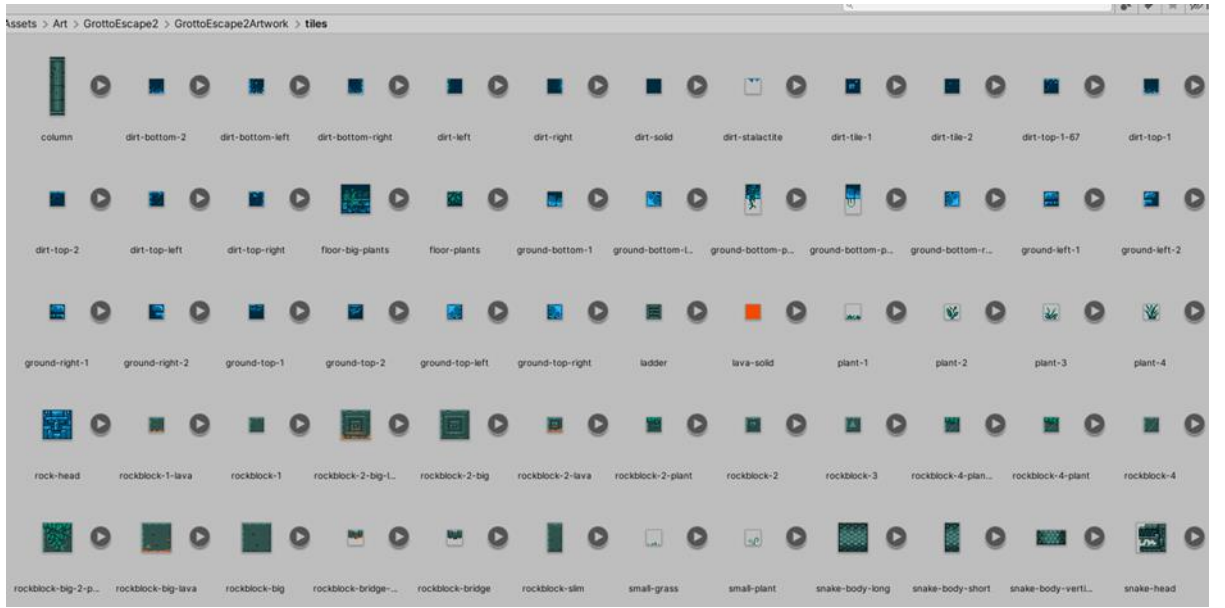


Şekil.3.1.3. 2D Pixel Item Paketi [8]



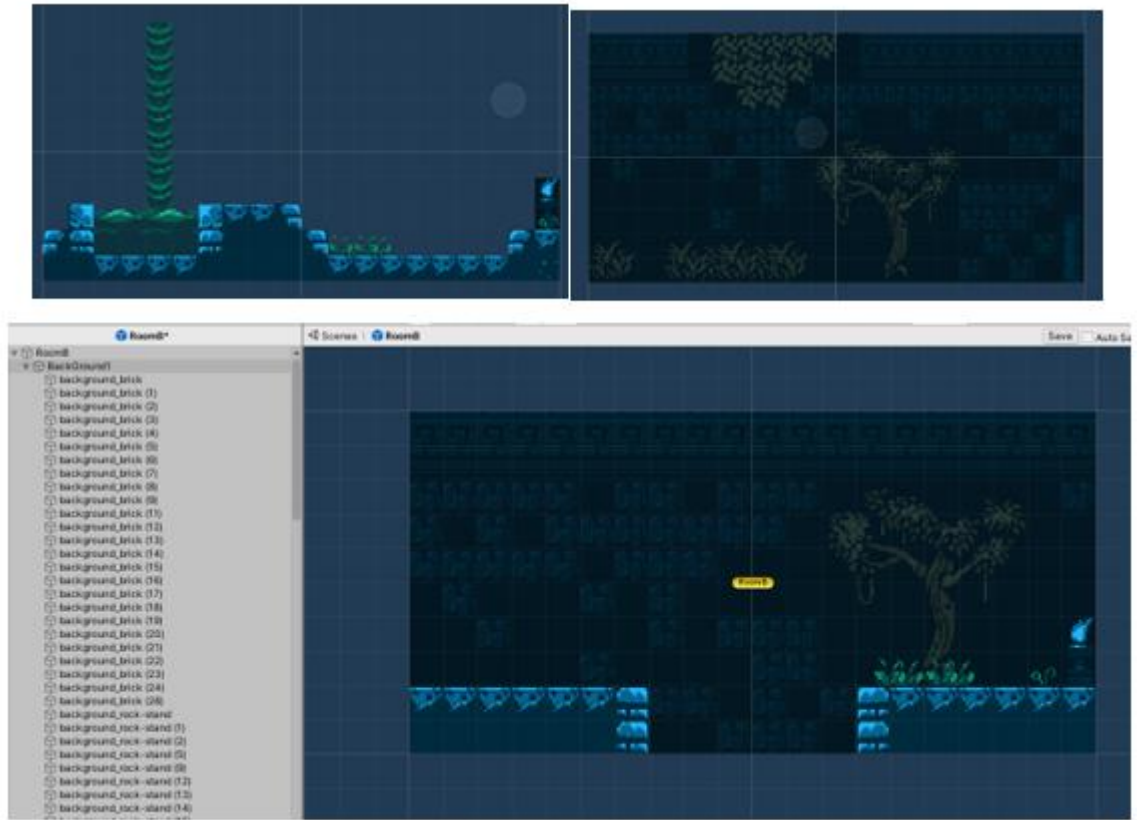
Şekil.3.1.4. Oyundaki Kalpler

Şekil.3.1.7 de 2D pixel Item paketinden alınmış kalpler görülür.



Şekil.3.2.5. Tiles

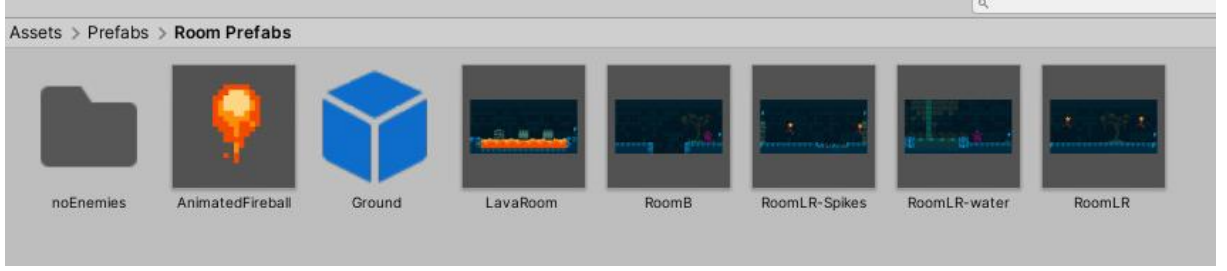
Grotto escape paketinde bulunan ve Şekil.3.2.5’ de görülen tile’lar ( küçük kare parçacıkları) kullanılarak Şekil.3.2.6 ‘ de görülen şekilde odalar yaratılmıştır.



Şekil.3.2.6.Odaların Yaratımı



### 3.3. LEVEL DESIGN



Şekil.3.3.1.Odacıklar

Level tasarımında Spelunky (2008) adlı oyundan esinlenilmiş ve procedural level generation kullanılmıştır.

Oyunun bir leveli 4\*3 , 12 odacıktan oluşur. Bu odacıklar her oyuna yeniden başlandığında belirli kurallar dahilinde rastgele bir şekilde dizilir. Böylece her seferinde benzersiz bir level tasarlanmış olur.

Level dizaynına oda tasarımı ile başlanmıştır. Odaların birleştirilebilmesi için her odanın sağ, solu ve üstü açıktır. Şekil.3.3.1.' te görülen 5 farklı tür oda tasarımı yapılmıştır. Daha sonra istenirse bu sayı arttırılabilir. Bu 5 farklı oda aşağısında açıklık bulunmasına göre iki türe ayrılabilirler.

RoomB ( Room Bottom ) aşağı çıkışı olan tek odadır.



Şekil.3.3.2.Odacık 1

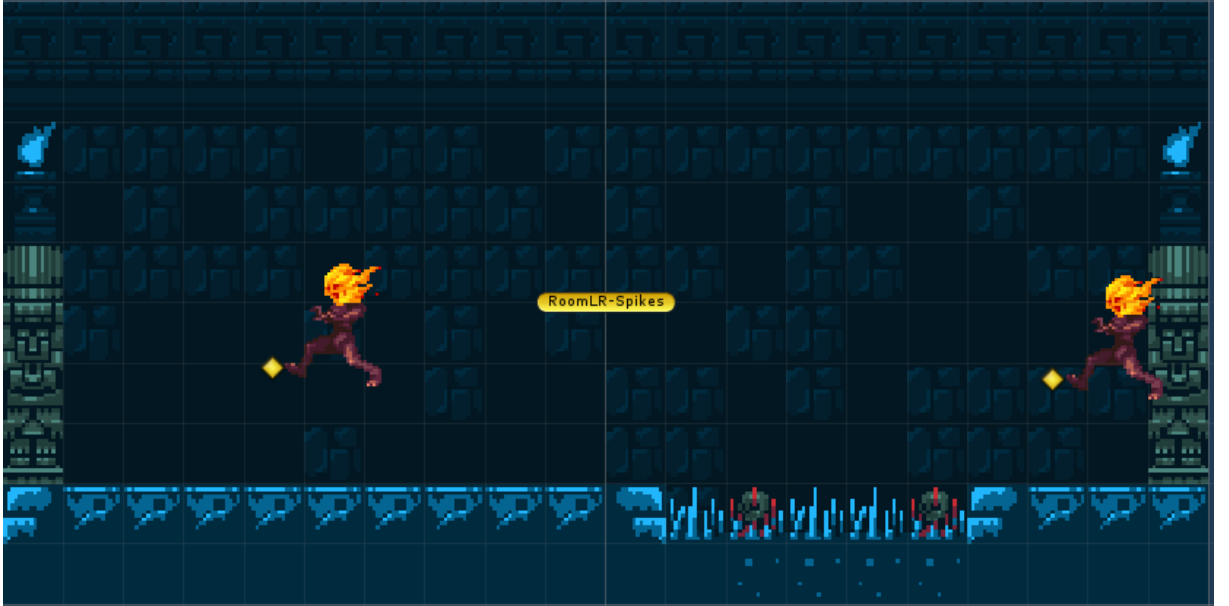


İkinci tür odalar aşağısı kapalı olan odalardır.



Şekil.3.3.3.Odacık 2

Room with spikes



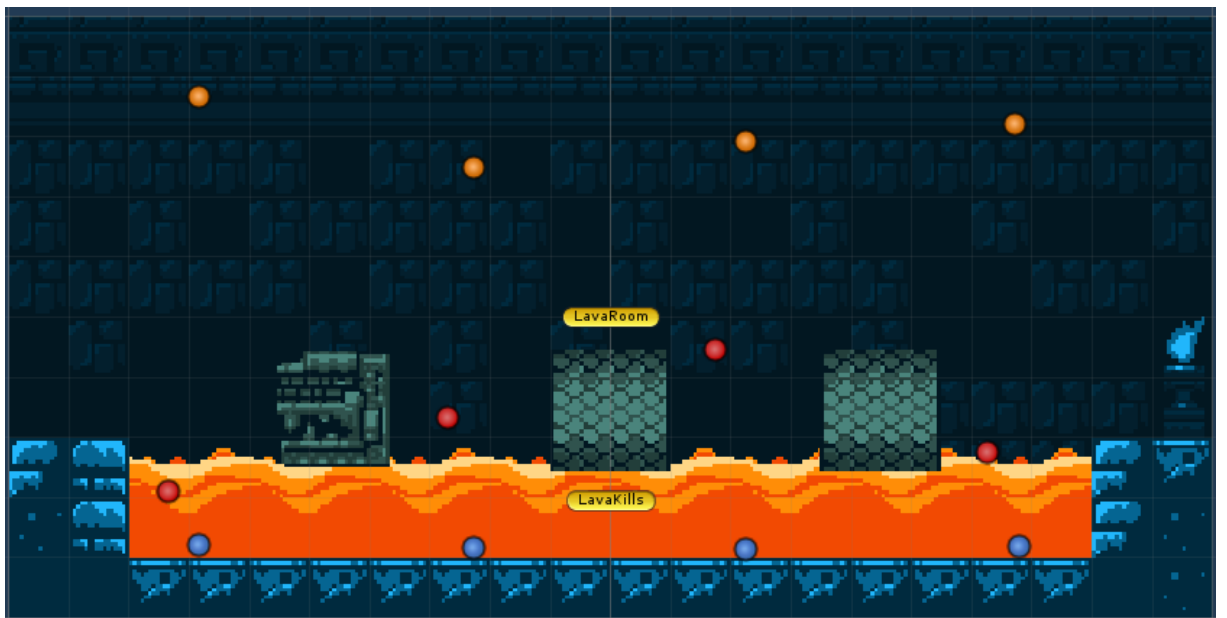
Şekil.3.3.4.Odacık 3

### Room with waterfall

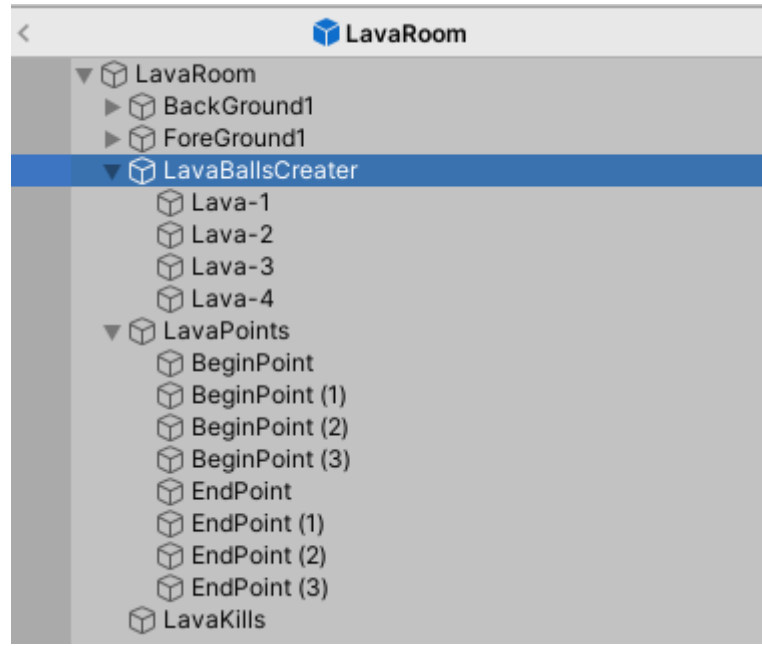


Şekil.3.3.5.Odacık 4

### Lava Room



Şekil.3.3.6.Odacık 5



Şekil.3.3.7.LavaRoom Prefabı

Şekil.3.3.6'daki LavaRoom Prefabına baktığımızda gördüğümüz kırmızı noktalar lava toplarının oluşturulacağı noktalardır. Mavi ve turuncu noktalar oluşan lava toplarının bu noktalar arasında gidip gelmesi için yerleştirilmişlerdir. Noktaların adlarını Şekil.3.3.7' de görebiliriz.

Şekil.3.3.8'de kırmızı noktalarda lava topları oluşturulması sağlayan kod görülmektedir.

```
LavaBall.cs  LavaController.cs*  LevelGeneration.cs  PlayerHealth.cs
Assembly-CSharp  LavaController

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  References
6  public class LavaController : MonoBehaviour
7  {
8      public Transform[] BeginPoints;
9      public Transform[] EndPoints;
10
11     public GameObject FireBallPrefab;
12
13     References
14     void Start()
15     {
16         //instantiate 4 fireballs in beginPoints
17         Instantiate(FireBallPrefab, BeginPoints[0].position , Quaternion.identity);
18         Instantiate(FireBallPrefab, BeginPoints[1].position , Quaternion.identity);
19         Instantiate(FireBallPrefab, BeginPoints[2].position, Quaternion.identity);
20         Instantiate(FireBallPrefab, BeginPoints[3].position, Quaternion.identity);
21
22     }
23
24
25
26
```

Şekil.3.3.8.LavaController kodu



```
4
5 public class LavaBall : MonoBehaviour
6 {
7     //direction
8     public int goingUp;
9
10    public float speed = 4;
11    public Rigidbody2D rb;
12
13    public int damage = 10;
14
15
16
17    void Start()
18    {
19        goingUp = 1;
20        Move();
21    }
22
23    private void Update()
24    {
25        Move();
26    }
27
28    private void OnTriggerEnter2D(Collider2D collision)
29    {
30        //change direction when colide with tag name firepoints
31        if (collision.CompareTag("FirePoints")) {
32            ChangeDirection();
33        }
34
35        else if (collision.CompareTag("Player")) {
36            collision.GetComponent<PlayerHealth>().playerAnim.SetTrigger("hurt");
37            collision.GetComponent<PlayerHealth>().TakeDamage(damage);
38            goingUp = -1;
39        }
40    }
41
42
43    void Move()
44    {
45        rb.velocity = transform.up * goingUp * speed;
46    }
47
48    void ChangeDirection()
49    {
50        goingUp = -1 * goingUp;
51    }
52
53 }
```

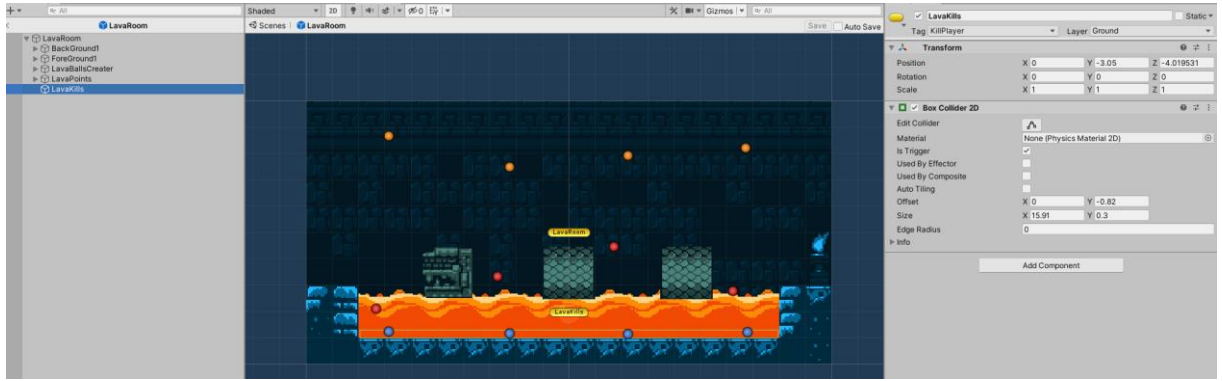
Şekil.3.3.9.LavaBall kodu

LavaBall kodu (Şekil.3.3.9’da görölür ) oluşturulan Lava toplarına eklenmiş bir koddur. Bu kod lava toplarının turuncu ve mavi noktalara değdiği zaman yön değıştirmesini ve aşağı yukarı yönde ilerlemesini sağlar. Bu noktalara ve lava topuna circle Collider 2d eklendiği için birbirilerine değdikleri zaman OnTriggerEnter2D fonksiyonu ile algılanırlar.

Oyuncu Şekil.3.3.10'da görülen SpikesRoom'da ve Şekil.3.3.11'de görülen LavaRoom' da dikenlere ve lavaya düştüğü zaman ölür. Buda collider eklenmesi ve Şekil.3.3.12' de görülen onTriggerEnter2D fonksiyonun Playerhealth cokuna eklenmesiyle yapılmıştır.(player health kodu bölüm 3.4 de incelenmiştir)



Şekil.3.3.10.Spikes Roomdaki dikenler



Şekil.3.3.11.LavaRoom daki lava

```
0 references
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("KillPlayer")) {
        playerAnim.SetTrigger("hurt");
        TakeDamage(100);
    }
}
```

Şekil.3.3.12.PlayerHealth kodu lava ve dikenlere çarpma durumu

```

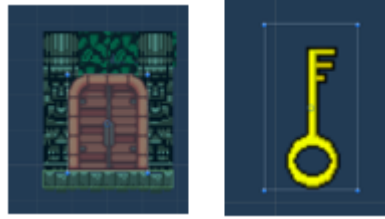
KeySpawner.cs  X Key.cs  Door.cs  LavaBall.cs  LavaController.cs*  LevelGeneration.cs  PlayerHead.cs
Assembly-CSharp
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class KeySpawner : MonoBehaviour
6  {
7      public GameObject key;
8      public GameObject[] whichRow; // row1 = 5f, row2 =-5f ,row3 =-15f, row4=-25f;
9      float randX;
10
11      Vector2 whereToSpawn;
12
13
14      void Start()
15      {
16          randX = Random.Range(0f, 60f);
17          int rand = Random.Range(0, 4);
18          whereToSpawn = new Vector2(randX, whichRow[rand].transform.position.y);
19
20          Instantiate(key, whereToSpawn, Quaternion.identity);
21      }
22
23
24
25

```

Şekil.3.3.14.KeySpawner kodu

Şekil.3.3.13' de görülen KeySpawner koduyla oyun her başladığında labirentin random bir noktasında anahtar yaratılır.

Oyuncunun bölümü geçebilmesi için anahtarı bulup kapıyı açması gerekmektedir. Şekil.3.3.15'de kapı ve anahtar görülür.



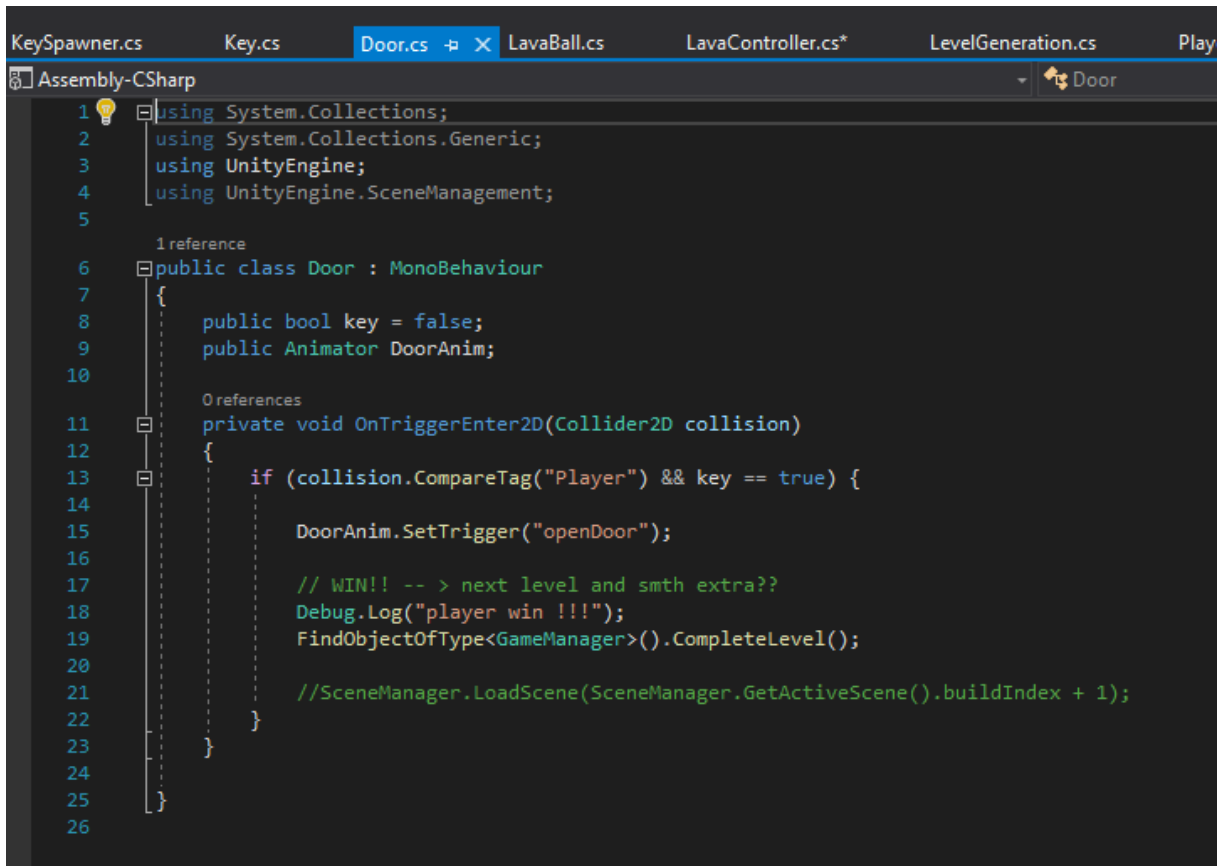
Şekil.3.3.15. Kapı ve Anahtar

```

KeySpawner.cs  X Key.cs  X Door.cs  LavaBall.cs  LavaController.cs*
Assembly-CSharp
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Key : MonoBehaviour
6  {
7      private void OnTriggerEnter2D(Collider2D collision)
8      {
9          if (collision.CompareTag("Player")) {
10
11              // open the door when the key is found
12              FindObjectOfType<Door>().key = true;
13              Destroy(gameObject);
14          }
15      }
16
17

```

Şekil.3.3.16.Key kodu



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.SceneManagement;
5
6 public class Door : MonoBehaviour
7 {
8     public bool key = false;
9     public Animator DoorAnim;
10
11     private void OnTriggerEnter2D(Collider2D collision)
12     {
13         if (collision.CompareTag("Player") && key == true) {
14
15             DoorAnim.SetTrigger("openDoor");
16
17             // WIN!! -- > next level and smth extra??
18             Debug.Log("player win !!!");
19             FindObjectOfType<GameManager>().CompleteLevel();
20
21             //SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
22         }
23     }
24 }
25
26
```

Şekil.3.3.17. Door Kodu

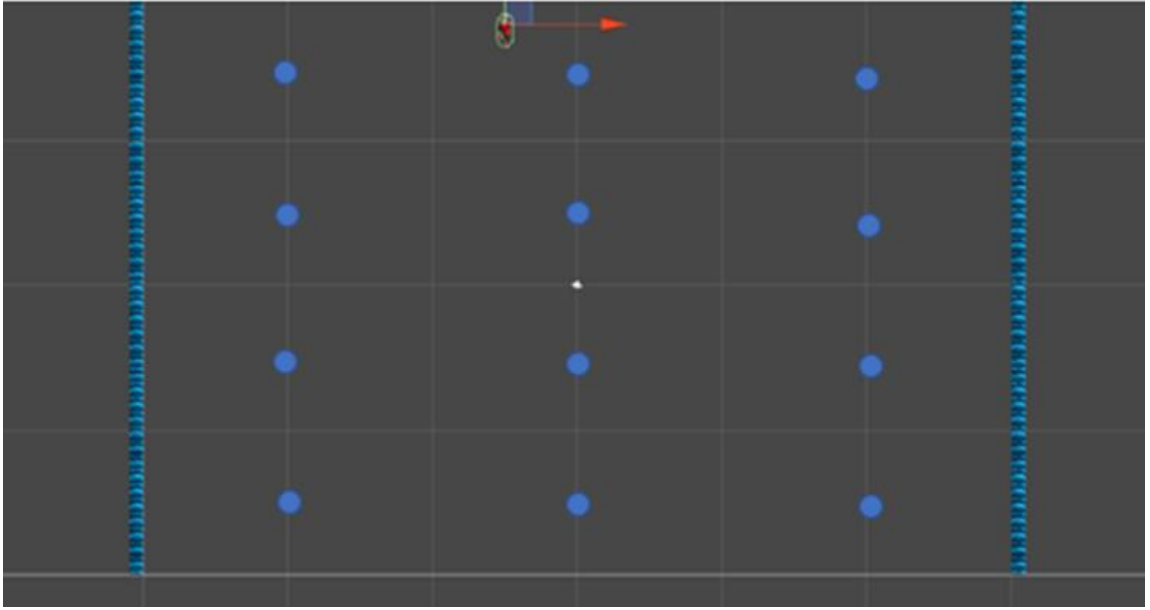
Şekil.3.3.16'daki Key koduyla oyuncu anahtarı bulduğu zaman ( anahtara dokunduğu zaman ), Şekil.3.3.17'deki Door kodundaki anahtar bool değişkeni true yapılır ve anahtar sahneden silinir.

Door kodunda key adlı bir bool değişkeni vardır. Bu değişken başlangıçta false dur. Oyuncu anahtarı bulduğu zaman bu değişken true olur.

Oyuncu anahtarı bulur ve kapıya gelirse kapı açılır. ( kapı açılma animasyonu oynatılır. )

Daha sonra GameManager kodundaki CompleteLevel fonksiyonu çağrılır. Buda bu bölümü bitirir ve yeni sahneyi yükler.

Eğer oyuncu anahtarı bulmadan kapıya gelirse kapı açılmaz.



Şekil.3.3.18.Oda Pozisyonları

Level dizaynında 12 oda belirli kurallar dahilinde Şekil.3.3.18’deki oda pozisyonlarına dizilir.

1. İlk önce, ilk yatay sıradaki üç pozisyondan birisi rastgele bir şekilde seçilir. Bu pozisyona yeni bir rastgele oda (lava room dışında) konulur. Gidiş yönü (yeni odanın konulacağı yer) (sağ , sol , aşağı) rastgele bir şekilde seçilir.
2. Eğer seçilen yön aşağı ise bu pozisyondaki oda silinir ve aşağı açıklığı olan oda bu pozisyona konulur. Eğer seçilen yol sağ veya solsa bir değişiklik yapılmaz. Daha sonra seçilen yönde ilerlenir. Bu pozisyona yeni bir rastgele oda konulur ve yeni bir yön belirlenir.
3. 2. adım çözüm yolu üretimi bitene kadar devam eder. Çözüm yolu üretiminin durması dördüncü sıraya gelindiğinde; yeni yön aşağı olursa veya en soldayken yeni yol sol olursa veya en sağdayken yeni yol sağ olursa sağlanır. Şekil.3.3.19’da soldaki resimde çözüm yolu görülür.
4. Daha sonra çözüm yolu dışında kalan pozisyonlara rastgele odalar (lava room dahil) atanır. Şekil.3.3.19’da sağdaki resimde çözüm yolu dışında kalan odalar görülür

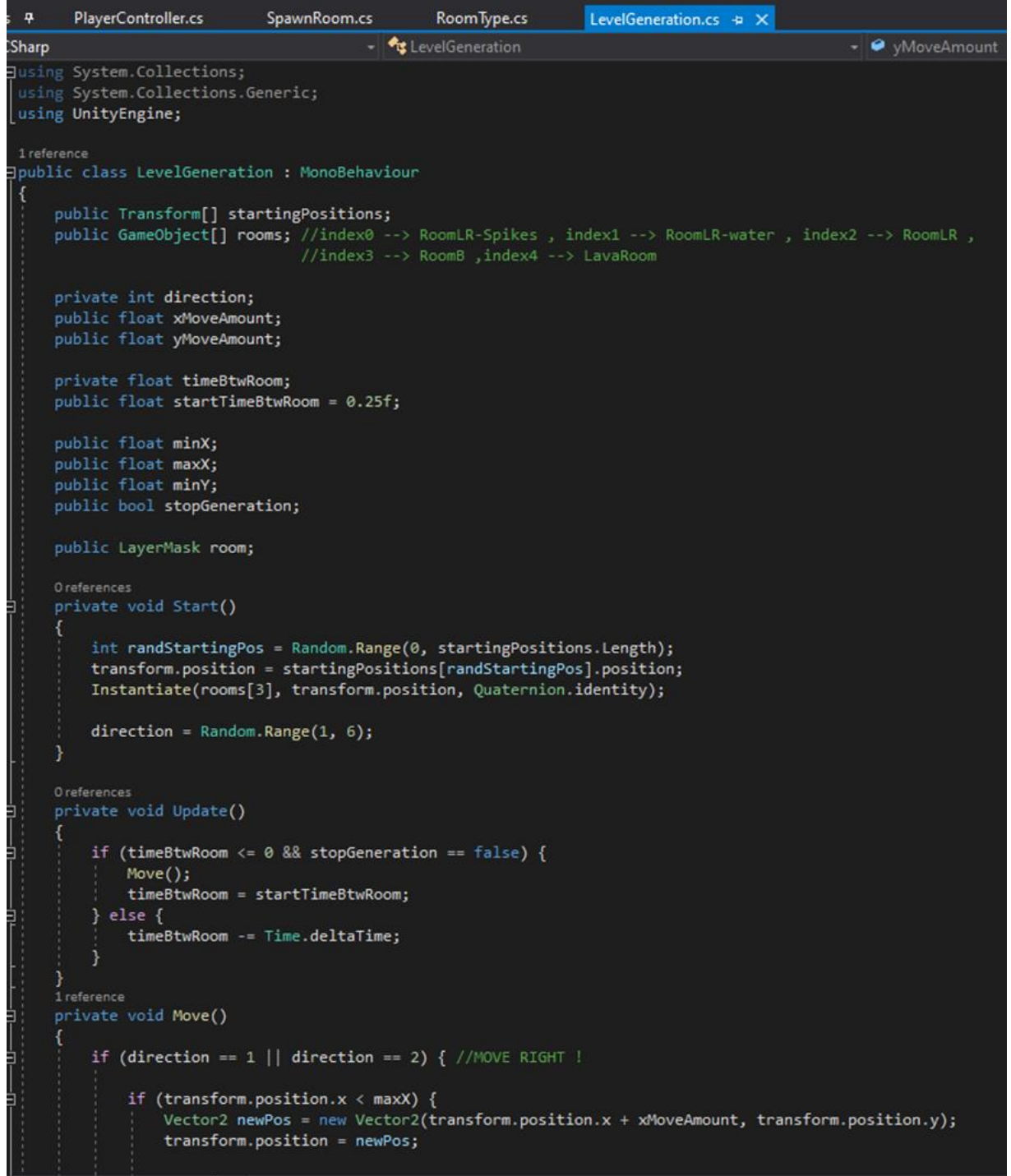


Şekil.3.3.19. Çözüm Yolu Ve Çözüm Yolu Dışında Kalan Odalar



Odacıkların kullanılarak labirentin oluşturulmasında Şekil.3.3.20 ve Şekil.3.3.21’de görülen LevelGenaration kodu , Şekil.3.3.23’de.RoomType kodu ve Şekil.3.3.24’de SpawnRoom kodu kullanılmıştır. Bunlar aşağıda verilmiştir.

LevelGeneration kodunda Solution Path oluşturulur.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

1 reference
public class LevelGeneration : MonoBehaviour
{
    public Transform[] startingPositions;
    public GameObject[] rooms; //index0 --> RoomLR-Spikes , index1 --> RoomLR-water , index2 --> RoomLR ,
                                //index3 --> RoomB ,index4 --> LavaRoom

    private int direction;
    public float xMoveAmount;
    public float yMoveAmount;

    private float timeBtwRoom;
    public float startTimeBtwRoom = 0.25f;

    public float minX;
    public float maxX;
    public float minY;
    public bool stopGeneration;

    public LayerMask room;

    0 references
    private void Start()
    {
        int randStartingPos = Random.Range(0, startingPositions.Length);
        transform.position = startingPositions[randStartingPos].position;
        Instantiate(rooms[3], transform.position, Quaternion.identity);

        direction = Random.Range(1, 6);
    }

    0 references
    private void Update()
    {
        if (timeBtwRoom <= 0 && stopGeneration == false) {
            Move();
            timeBtwRoom = startTimeBtwRoom;
        } else {
            timeBtwRoom -= Time.deltaTime;
        }
    }

    1 reference
    private void Move()
    {
        if (direction == 1 || direction == 2) { //MOVE RIGHT !

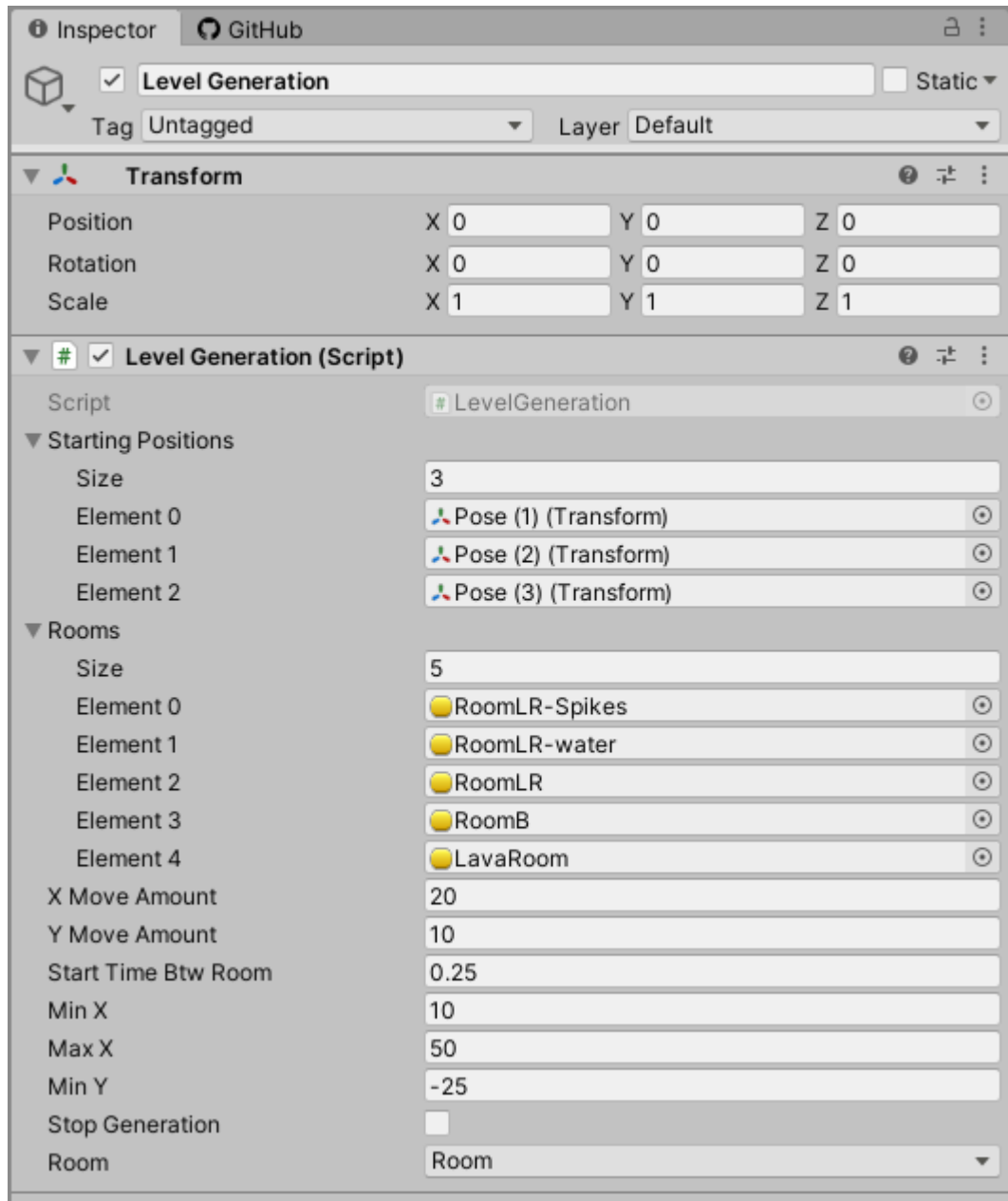
            if (transform.position.x < maxX) {
                Vector2 newPos = new Vector2(transform.position.x + xMoveAmount, transform.position.y);
                transform.position = newPos;
            }
        }
    }
}
```

Şekil.3.3.20.LevelGenaration kodu

```
PlayerAttack.cs  PlayerController.cs  SpawnRoom.cs  RoomType.cs  LevelGeneration.cs*  Move()
Assembly-CSharp  LevelGeneration

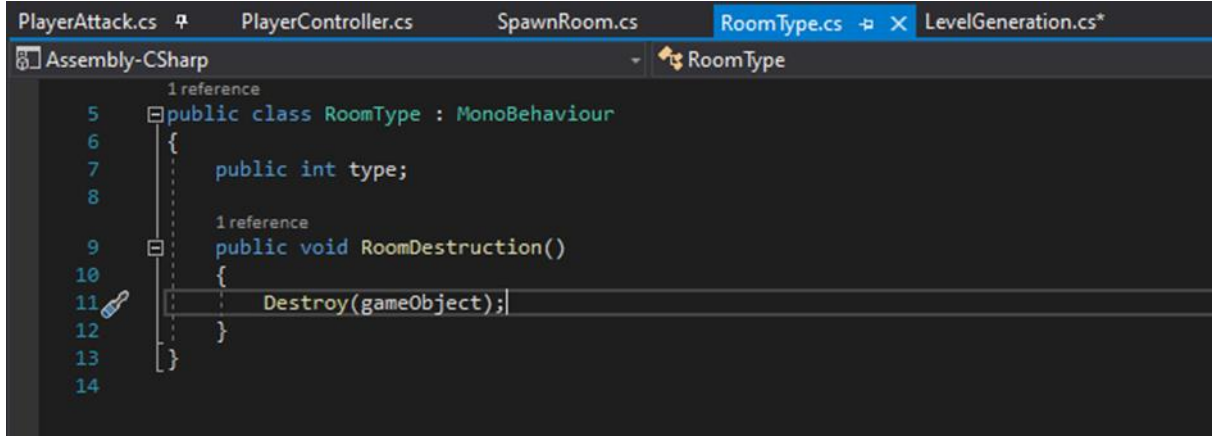
51  int rand = Random.Range(0, 3);
52  Instantiate(rooms[rand], transform.position, Quaternion.identity);
53
54  direction = Random.Range(1, 6);
55  if (direction == 3) {
56      direction = 2;
57  } else if (direction == 4) {
58      direction = 5;
59  }
60
61  } else {
62      direction = 5;
63  }
64
65
66  } else if (direction == 3 || direction == 4) { //MOVE LEFT !
67      if (transform.position.x > minX) {
68          Vector2 newPos = new Vector2(transform.position.x - xMoveAmount, transform.position.y);
69          transform.position = newPos;
70
71          int rand = Random.Range(0, 3);
72          Instantiate(rooms[rand], transform.position, Quaternion.identity);
73          direction = Random.Range(3, 6);
74
75      } else {
76          direction = 5;
77      }
78
79  } else if (direction == 5) { //MOVE DOWN !
80      if (transform.position.y > minY) {
81
82          //replacing the room with an bottom room (index 3) before moving down
83          Collider2D roomDetection = Physics2D.OverlapCircle(transform.position, 1, room);
84          roomDetection.GetComponent<RoomType>().RoomDestruction();
85          Instantiate(rooms[3], transform.position, Quaternion.identity);
86
87          // MOVING DOWN
88          Vector2 newPos = new Vector2(transform.position.x, transform.position.y - yMoveAmount);
89          transform.position = newPos;
90          int rand = Random.Range(0, 3);
91          Instantiate(rooms[rand], transform.position, Quaternion.identity);
92
93          //NEXT DIRECTION
94          direction = Random.Range(1, 6);
95
96      } else {
97          //STOP LEVEL GENERATION
98          stopGeneration = true;
99      }
100
101  }
102
103
104
```

Şekil.3.3.21.LevelGenaration kodu devamı



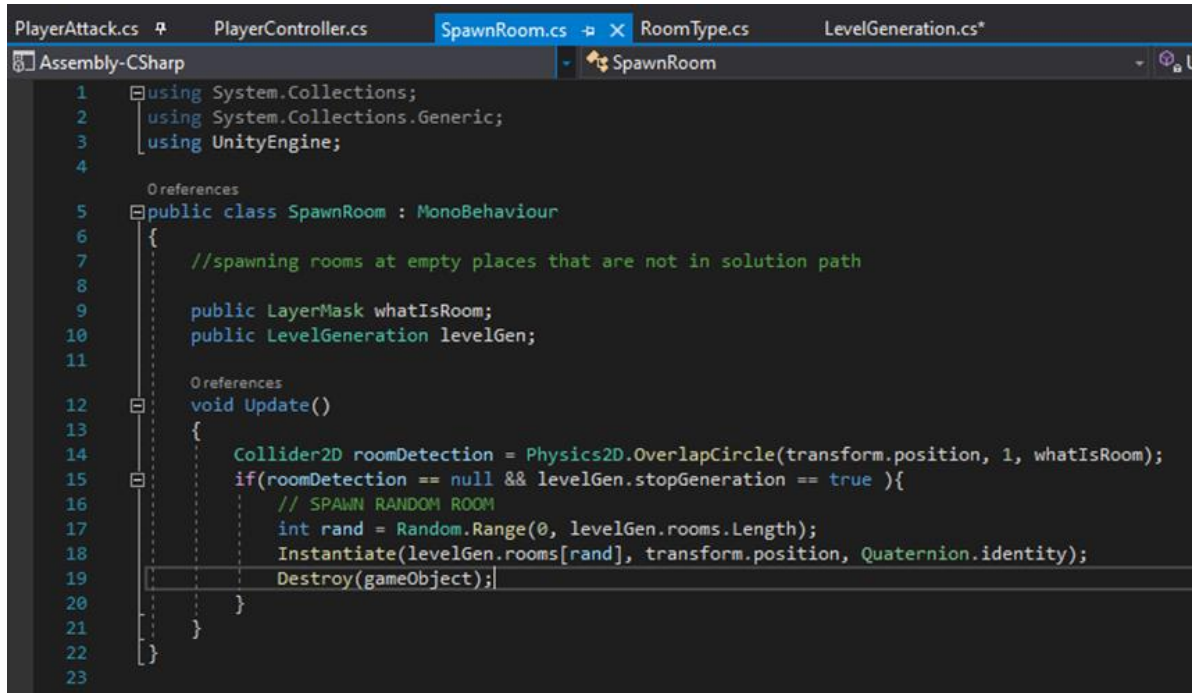
Şekil.3.3.22

Kod Unity oyun motorunda Şekil.3.3.22'deki gibi görünür. Buraya başlangıç pozisyonları ve odaların prefabları eklenmiştir.



Şekil.3.3.23.RoomType kodu

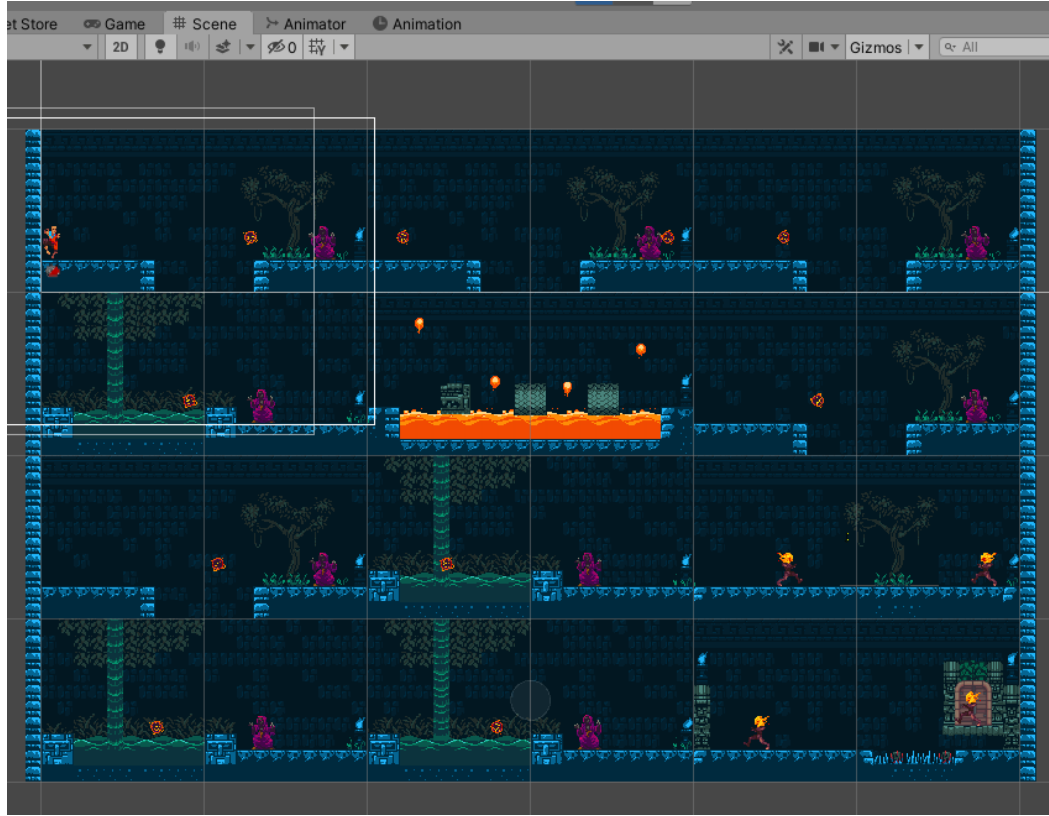
RoomType kodu üretilen odanın gerekli durumlarda silinmesi için kullanılır. Bu kod odalara eklendiği için gerekliliği olduğu zaman Level Generation tarafından çağrıldığında bağlı olduğu oda yok edilir.



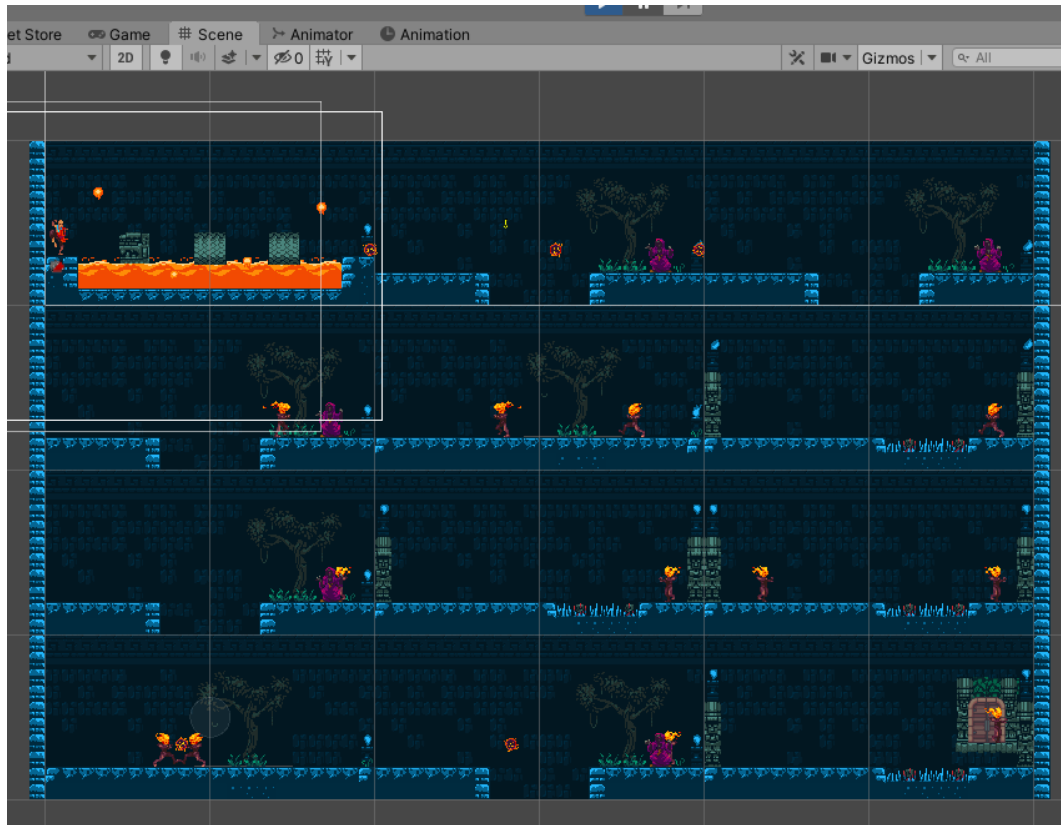
Şekil.3.3.24.SpawnRoom kodu

SpawnRoom kodu Solution Path oluşturulduktan sonra bu path üzerinde olmayan odaların üretilmesinde kullanılır.

## RASTGELE YARATILMIŞ LEVEL ÖRNEKLERİ

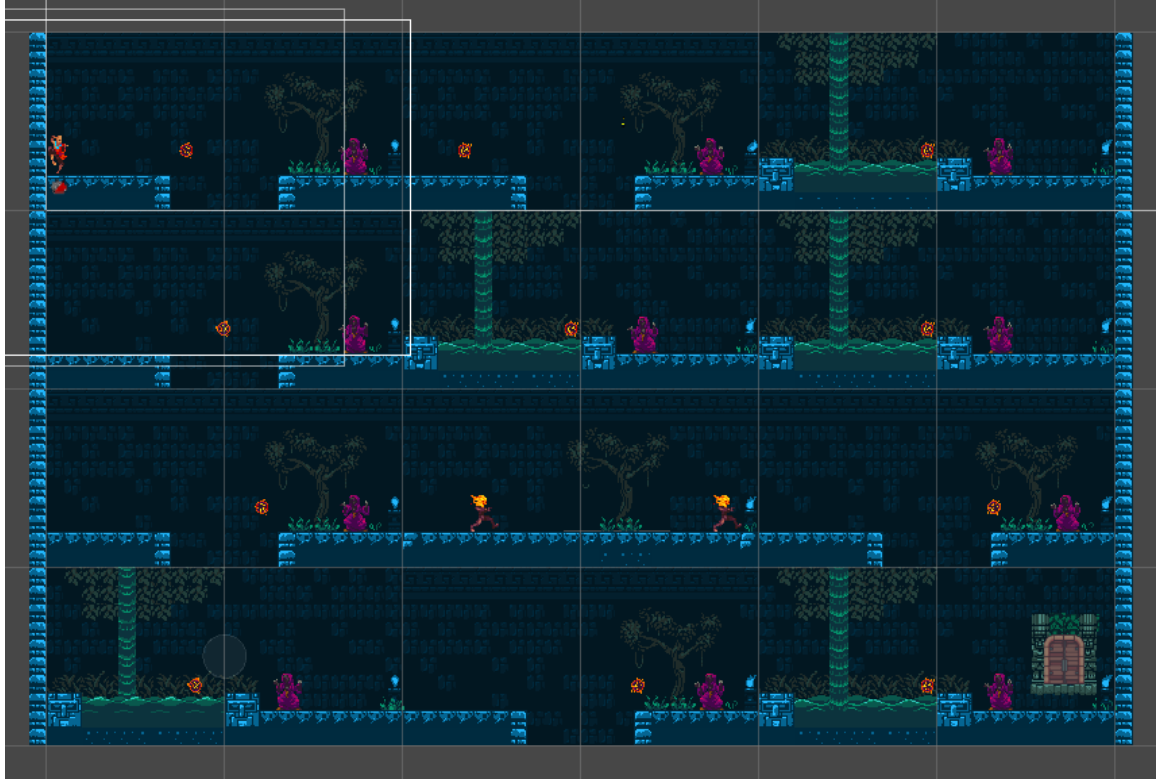


Şekil.3.3.25.Level örneği 1

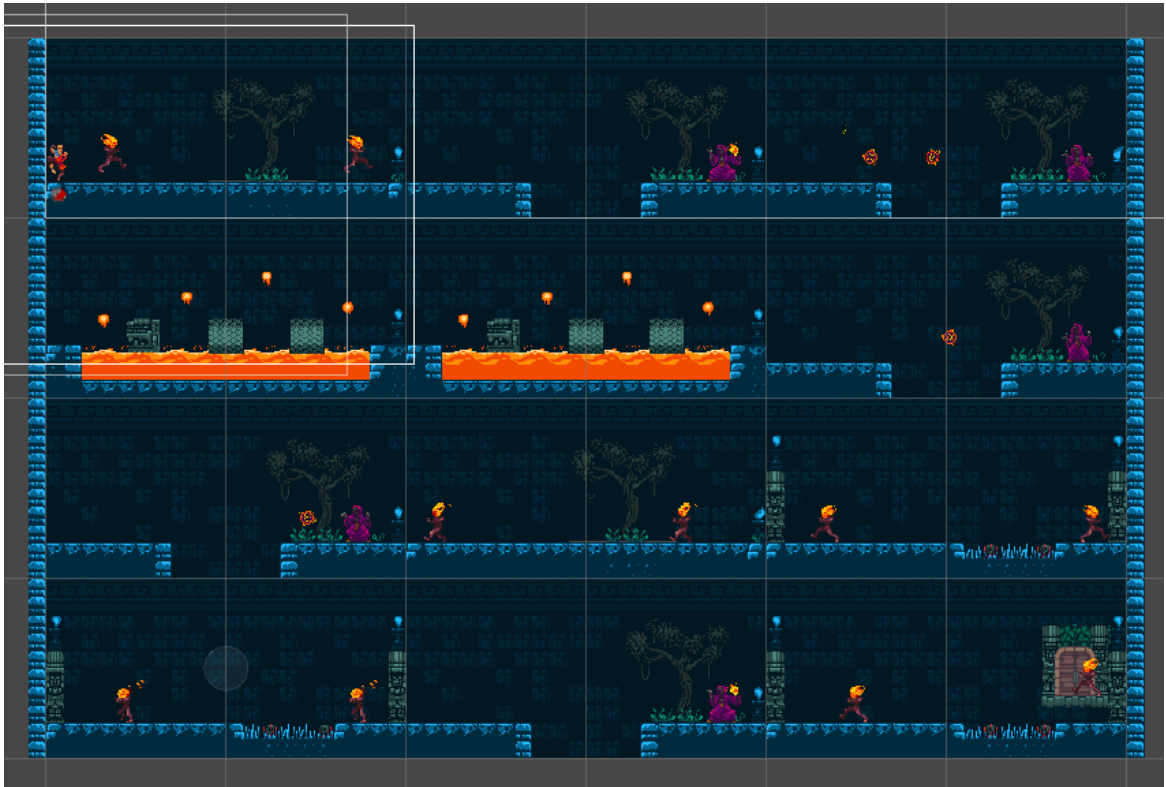


Şekil.3.3.26.Level örneği 2



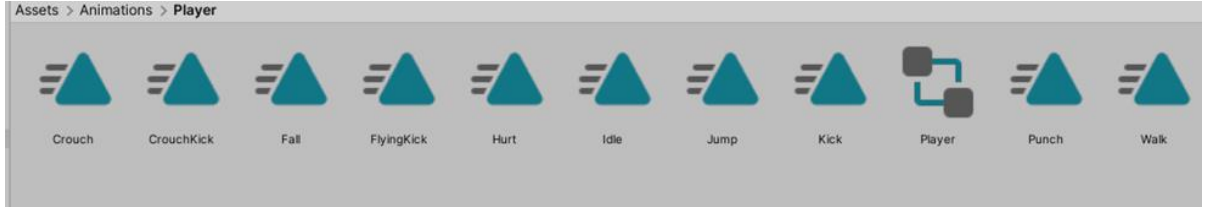


Şekil.3.3.27.Level örneği 3

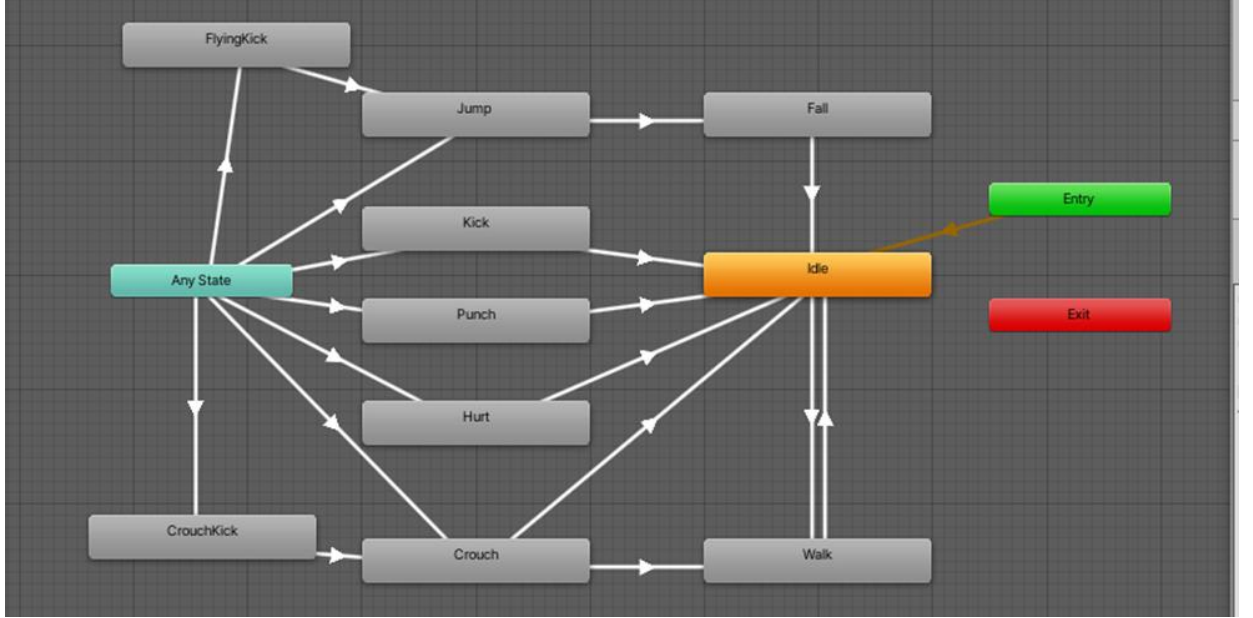


Şekil.3.3.27.Level örneği 4

### 3.4. OYUNCUNUN HAREKET VE KONTROLÜNÜN OLUŞTURULMASI



Şekil.3.4.1.Oyuncu Animasyonları



Şekil.3.4.2.Oyuncu Animator Stateleri

Layers	Parameters
	🔍 Name + ▾
isWalking	<input type="checkbox"/>
isJumping	<input type="checkbox"/>
isGrounded	<input type="checkbox"/>
isCrouching	<input type="checkbox"/>
Punch	<input checked="" type="radio"/>
Kick	<input checked="" type="radio"/>
FlyingKick	<input checked="" type="radio"/>
CrouchKick	<input checked="" type="radio"/>
hurt	<input checked="" type="radio"/>

Şekil.3.4.3.Oyuncu Animasyon parametreleri

Oyuncunun hareketleri animasyonlarla ve fiziksel olarak yer değiştirmesiyle sağlanır. Animasyonlar Animator Controller ve kod ile kontrol edilirken fiziksel olarak yer değiştirme sadece kod ile kontrol edilir. Animator controller' da bulunan Şekil.3.4.3'de görülen parametreler true oldukları zaman ilgili animasyon oynar . Parametrelerin değerleri kodlarla kontrol edilir.

Oyuncunun kontrolü için iki kod kullanılmıştır.

- PlayerController ( şekiller 3.4.4 ile 3.4.6 arası gösterilmiştir.)
- PlayerAttack ( şekil 3.4.7 ve 3.4.8’de gösterilmiştir)

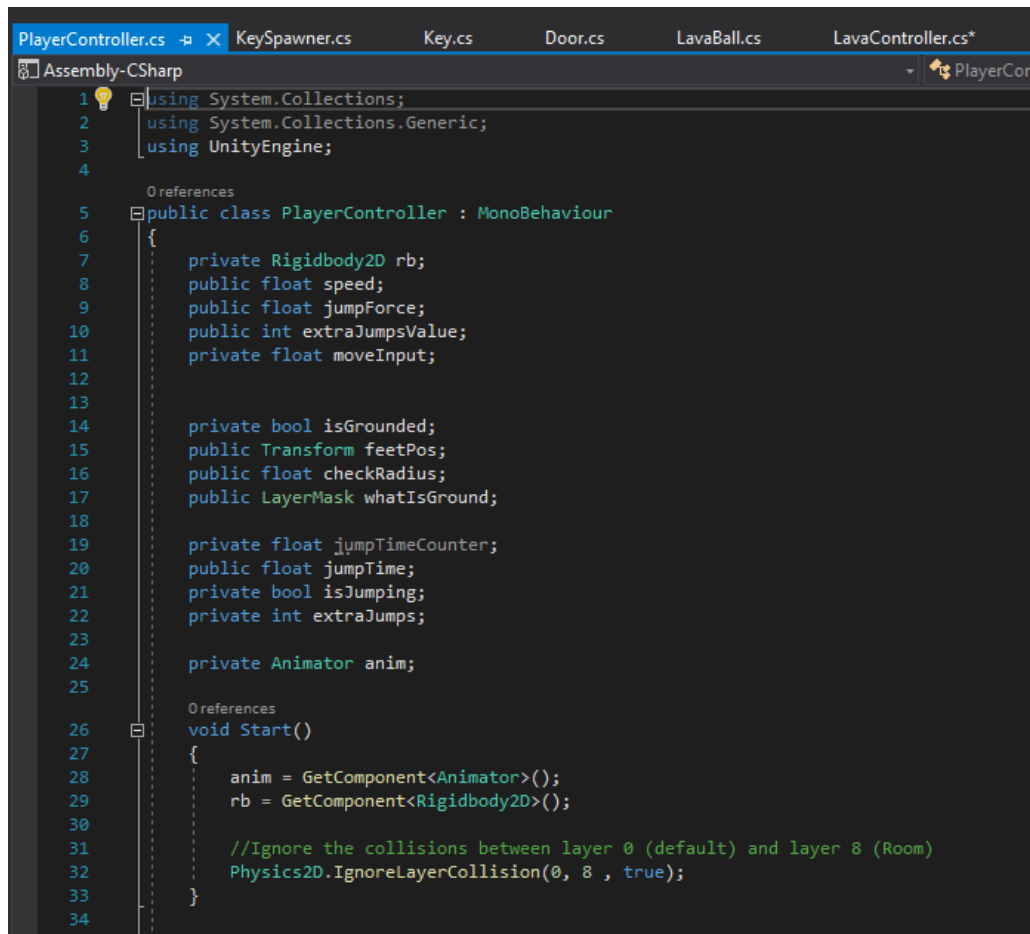
PlayerController kodunda oyuncunun fiziksel olarak yerdeğiřtirmesi kontrol edilmiştir. Zıpladığı zaman yukarı yönde , yürüdüğü zaman yürüdüğü yönde ilerlemesi sağlanmıştır.

Ayrıca PlayerController kodunda oyuncunun Walking , jumping ve crouching animasyonları kontrol edilmiştir. Bu kontrol animator Controller da bulunan parametrelere true , false verilerek sağlanmıştır.

PlayerController koduyla oyuncu ok tuşlarını kullanarak sağa ve sola doğru yürüyebilir , zıplayabilir ve çömelebilir.

Input.GetKey kullanılarak basılan tuşlar algılanır. İstenilen tuşa basıldığında animator controllerda ilgili parametre true olur ve animasyon oynar.

Oyuncunun fiziksel olarak yerdeğiřtirmesi velocity’i klavyeden alınan inputla update edilmesiyle sağlanmıştır. Oyuncu ok tuşuna bastığı sürece o yönde ilerler.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerController : MonoBehaviour
6 {
7     private Rigidbody2D rb;
8     public float speed;
9     public float jumpForce;
10    public int extraJumpsValue;
11    private float moveInput;
12
13
14    private bool isGrounded;
15    public Transform feetPos;
16    public float checkRadius;
17    public LayerMask whatIsGround;
18
19    private float jumpTimeCounter;
20    public float jumpTime;
21    private bool isJumping;
22    private int extraJumps;
23
24    private Animator anim;
25
26    void Start()
27    {
28        anim = GetComponent<Animator>();
29        rb = GetComponent<Rigidbody2D>();
30
31        //Ignore the collisions between layer 0 (default) and layer 8 (Room)
32        Physics2D.IgnoreLayerCollision(0, 8 , true);
33    }
34
35
```

Şekil.3.4.4.Player Controller kodu



```

35
36 void FixedUpdate()
37 {
38     moveInput = Input.GetAxis("Horizontal");
39     rb.velocity = new Vector2(moveInput * speed, rb.velocity.y);
40 }
41
42
43 references
44 private void Update()
45 {
46     isGrounded = Physics2D.OverlapCircle(feetPos.position, checkRadius, whatIsGround);
47     anim.SetBool("isGrounded", isGrounded);
48
49     //for animations
50     if (moveInput == 0 || isGrounded == false) {
51         anim.SetBool("isWalking", false);
52     } else {
53         anim.SetBool("isWalking", true);
54     }
55
56     //for making character look right and left side
57     if (moveInput > 0) {
58         transform.eulerAngles = new Vector3(0, 0, 0);
59     } else if (moveInput < 0) {
60         transform.eulerAngles = new Vector3(0, 180, 0);
61     }
62
63     // JUMP + DOUBLE JUMP
64     if (isGrounded == true) {
65         extraJumps = extraJumpsValue;
66     }
67     if (Input.GetKeyDown(KeyCode.UpArrow) && extraJumps > 0) {
68         isJumping = true;
69         anim.SetBool("isJumping", true);
70         rb.velocity = Vector2.up * jumpForce;
71         extraJumps--;
72     }
73     else if (Input.GetKeyDown(KeyCode.UpArrow) && extraJumps == 0 && isGrounded == true) {
74         isJumping = true;
75         anim.SetBool("isJumping", true);
76         rb.velocity = Vector2.up * jumpForce;
77     }
78
79     if (Input.GetKeyUp(KeyCode.UpArrow)) {
80         isJumping = false;
81     }
82
83     if (isJumping == false) {
84         anim.SetBool("isJumping", false);
85     }

```

Şekil.3.4.5.Player Controller kodu devamı

```

86
87
88 // CROUCH
89 if (Input.GetKeyDown(KeyCode.DownArrow) && isGrounded == true) {
90     anim.SetBool("isCrouching", true);
91 }
92 if (Input.GetKeyUp(KeyCode.DownArrow) || moveInput != 0) {
93     anim.SetBool("isCrouching", false);
94 }
95
96 if (rb.position.y < -31f) {
97     FindObjectOfType<GameManager>().EndGame();
98 }
99

```

Şekil.3.4.6.Player Controller kodu devamı

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerAttack : MonoBehaviour
6 {
7     public Animator playerAnim;
8
9     private float timeBtwAttack;
10    public float startTimeBtwAttack;
11
12    public Transform attackPoint;
13    public float attackRange = 0.5f;
14    public int attackDamage = 40;
15
16    public LayerMask enemyLayer;
17
18
19
20
21    private void Update()
22    {
23        if(timeBtwAttack <= 0) {
24            //then you can attack
25            if (Input.GetKey(KeyCode.Alpha1)) {
26                playerAnim.SetTrigger("Punch");
27                Attack();
28            } else if (Input.GetKey(KeyCode.Alpha2)) {
29
30
31                if (Input.GetKey(KeyCode.DownArrow) && playerAnim.GetBool("isGrounded")) {
32                    playerAnim.SetBool("isCrouching", false);
33                    playerAnim.SetTrigger("CrouchKick");
34                    Attack();
35                } else if(playerAnim.GetBool("isGrounded")== false) {
36                    playerAnim.SetTrigger("FlyingKick");
37                    Attack();
38                } else {
39                    playerAnim.SetTrigger("Kick");
40                    Attack();
41                }
42            }
43            timeBtwAttack = startTimeBtwAttack;
44        } else {
45            timeBtwAttack -= Time.deltaTime;
46        }
47    }
48
49
50
```

Şekil.3.4.7.Player Attack kodu

PlayerAttack kodunda oyuncunun saldırı animasyonlarının doğru tuşa basıldığında tetiklenmesi sağlanmıştır. Bu animasyonlarda parametreler trigger olarak kullanılmıştır.

Saldırı tuşları olarak 1 ve 2 tuşları seçilmiştir daha sonra bu tuşlar isteğe bağlı olarak değiştirilebilir.Eğer oyuncu 1 tuşuna basarsa yumruk atar. Eğer 2 tuşuna basarsa 3 değişik tekme atabilir.2 tuşuna bastığı zaman aynı zamanda çömeliyorsa yerden tekme ( CrouchKick ) atabilir.Eğer 2 tuşuna bastığı zaman yere dokunmuyorsa (isGrounded == false ) uçan tekme (FlyingKick ) atabilir.Eğer bu iki durumlar yoksa normal tekme atar.

```

4 references
void Attack()
{
    //Detect enemies in range of attack
    Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayer);

    //Damage them
    foreach(Collider2D enemy in hitEnemies) {
        enemy.GetComponent<Enemy>().TakeDamage(attackDamage);
    }
}

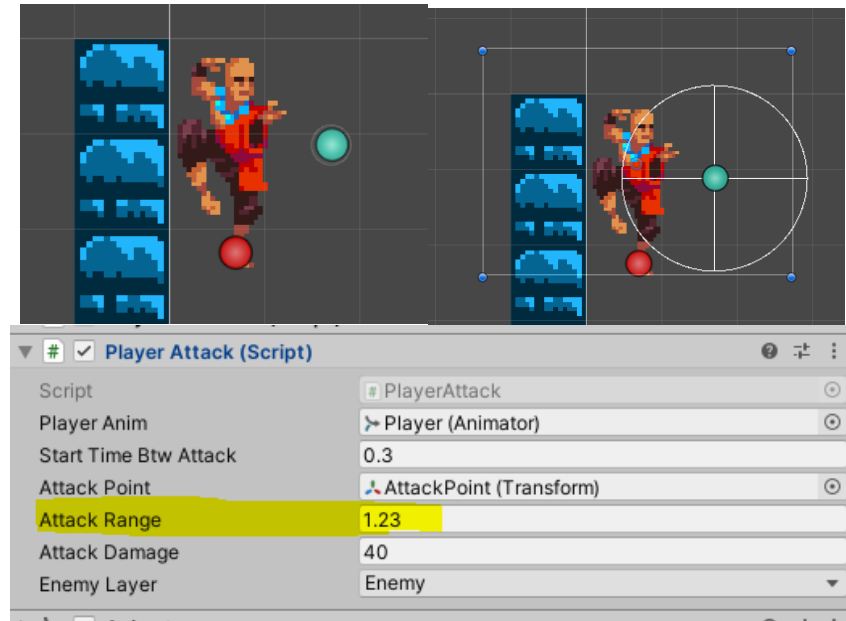
// to draw the attack range so we can change it as we like
0 references
private void OnDrawGizmosSelected()
{
    if (attackPoint == null)
        return;

    Gizmos.DrawWireSphere(attackPoint.position, attackRange);
}

```

Şekil.3.4.8.Player Attack kodu devamı

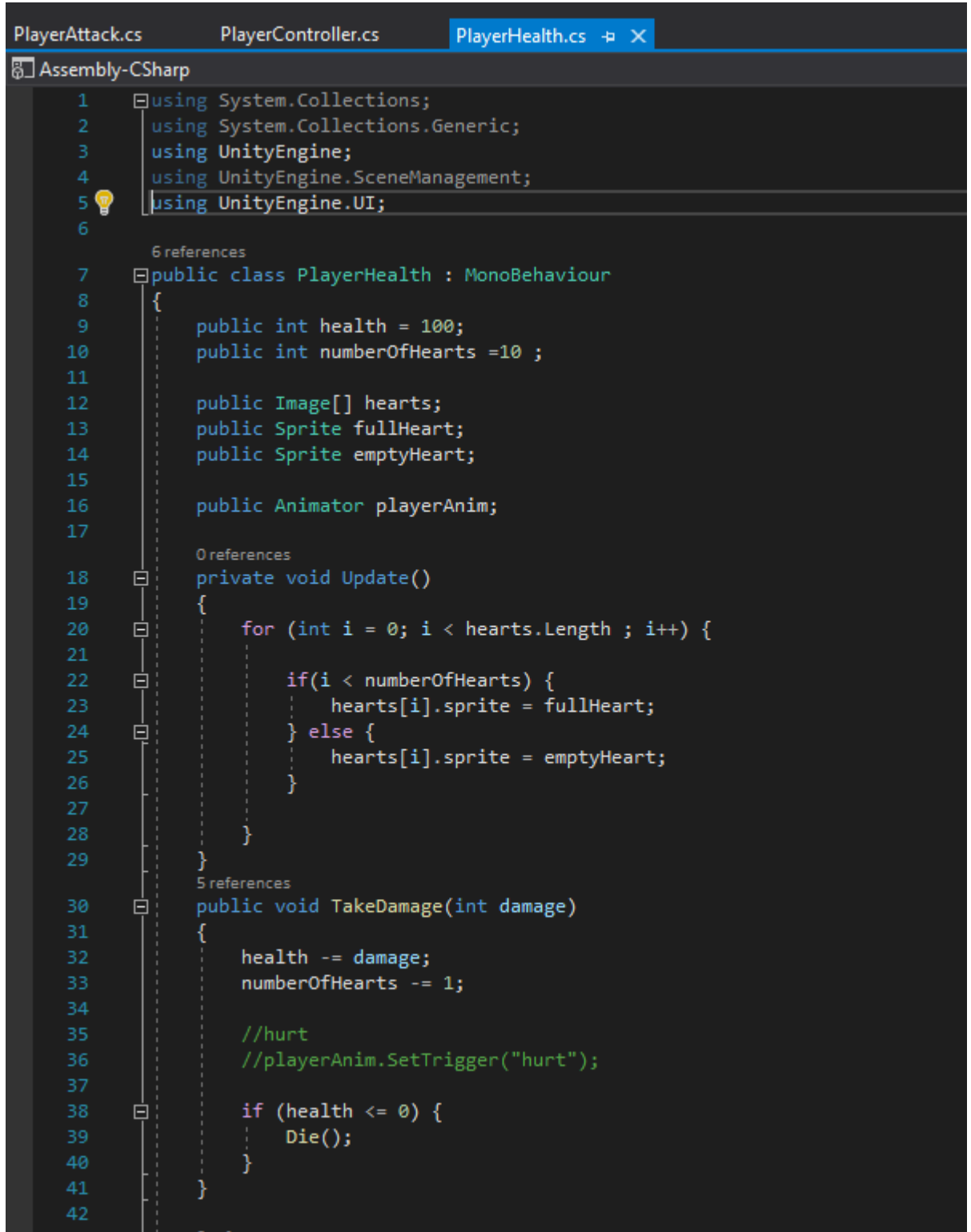
PlayerAttack kodunda bulunan Attack fonksiyonu ( şekil 3.4.8.'de görülür) oyuncunun düşmana saldırdığı zaman eğer düşmana değerse zarar vermesi için yazılmıştır.



Şekil.3.4.9.Player AttackPoint

Şekil.3.4.9'da görülen oyuncunun önünde bulunan mavi nokta attackPoint olup , 1.23 çapı vardır. Bu attackPoint merkezli ve 1.23 çapla oluşan çember oyuncunun saldırı alanıdır. Oyuncu saldırıda bulunduğu zaman bu alan içine giren düşman karakterlerin health scriptlerine ulaşarak onlara zarar verilir.

Oyuncunun düşmanlar tarafından saldırıya uğradığı zaman can kaybetmesi , lava ya ve dikenlere düştüğü zaman ölmesi için Şekil.3.4.10.'da görülen PlayerHealth kodu yazılmıştır.



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5  using UnityEngine.UI;
6
7  public class PlayerHealth : MonoBehaviour
8  {
9      public int health = 100;
10     public int numberOfHearts = 10 ;
11
12     public Image[] hearts;
13     public Sprite fullHeart;
14     public Sprite emptyHeart;
15
16     public Animator playerAnim;
17
18     private void Update()
19     {
20         for (int i = 0; i < hearts.Length ; i++) {
21             if(i < numberOfHearts) {
22                 hearts[i].sprite = fullHeart;
23             } else {
24                 hearts[i].sprite = emptyHeart;
25             }
26         }
27     }
28
29     public void TakeDamage(int damage)
30     {
31         health -= damage;
32         numberOfHearts -= 1;
33
34         //hurt
35         //playerAnim.SetTrigger("hurt");
36
37         if (health <= 0) {
38             Die();
39         }
40     }
41
42 }
```

Şekil.3.4.10. Player Health kodu

Ayrıca bu kodda ekranda ne kadar canı olduğunu göstermesi için eklenen UI elementlerin yönetimi yapılır. Number ofHearts ekranda gösterilen kırmızı kalp sayısıdır. Hasar aldıkça kalp sayısı azaltılır.

```

40     }
41 }
42
43 0 references
44 private void OnTriggerEnter2D(Collider2D collision)
45 {
46     if (collision.CompareTag("KillPlayer")) {
47         playerAnim.SetTrigger("hurt");
48         TakeDamage(100);
49     }
50
51     if (collision.CompareTag("Enemy")) {
52         playerAnim.SetTrigger("hurt");
53         TakeDamage(10);
54     }
55 }
56
57 1 reference
58 void Die()
59 {
60     FindObjectOfType<GameManager>().EndGame();
61 }
62 }
63

```

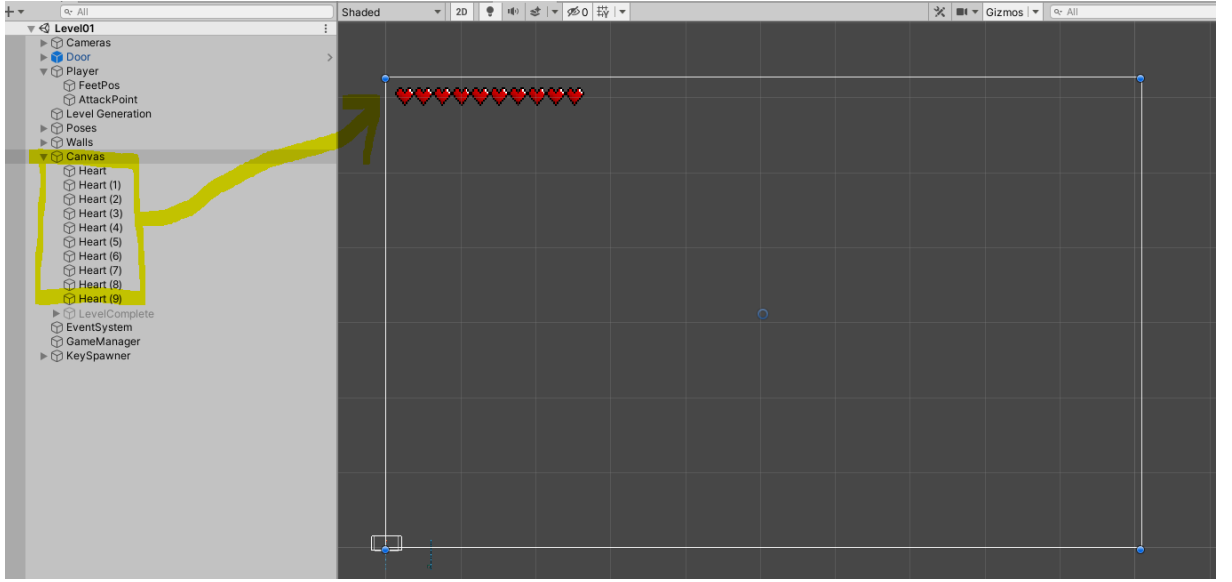
Şekil.3.4.11. Player Health kodu devamı

Oyuncu düşmana çarptığı zaman 10 hasar alır.

Oyuncu dikenlere veya lavaya düşünce bu objelerde KillPlayer tag i olduğu için 100 hasar alır ve toplam sağlık puanı 100 olduğu için ölür.Oyuncu öldüğü zaman oyuna yeniden başlar.



Şekil.3.4.12. PlayerHealth koduna kalp resimleri eklenmiştir.



Şekil.3.4.13.Oyuncunun Canını Gösteren UI

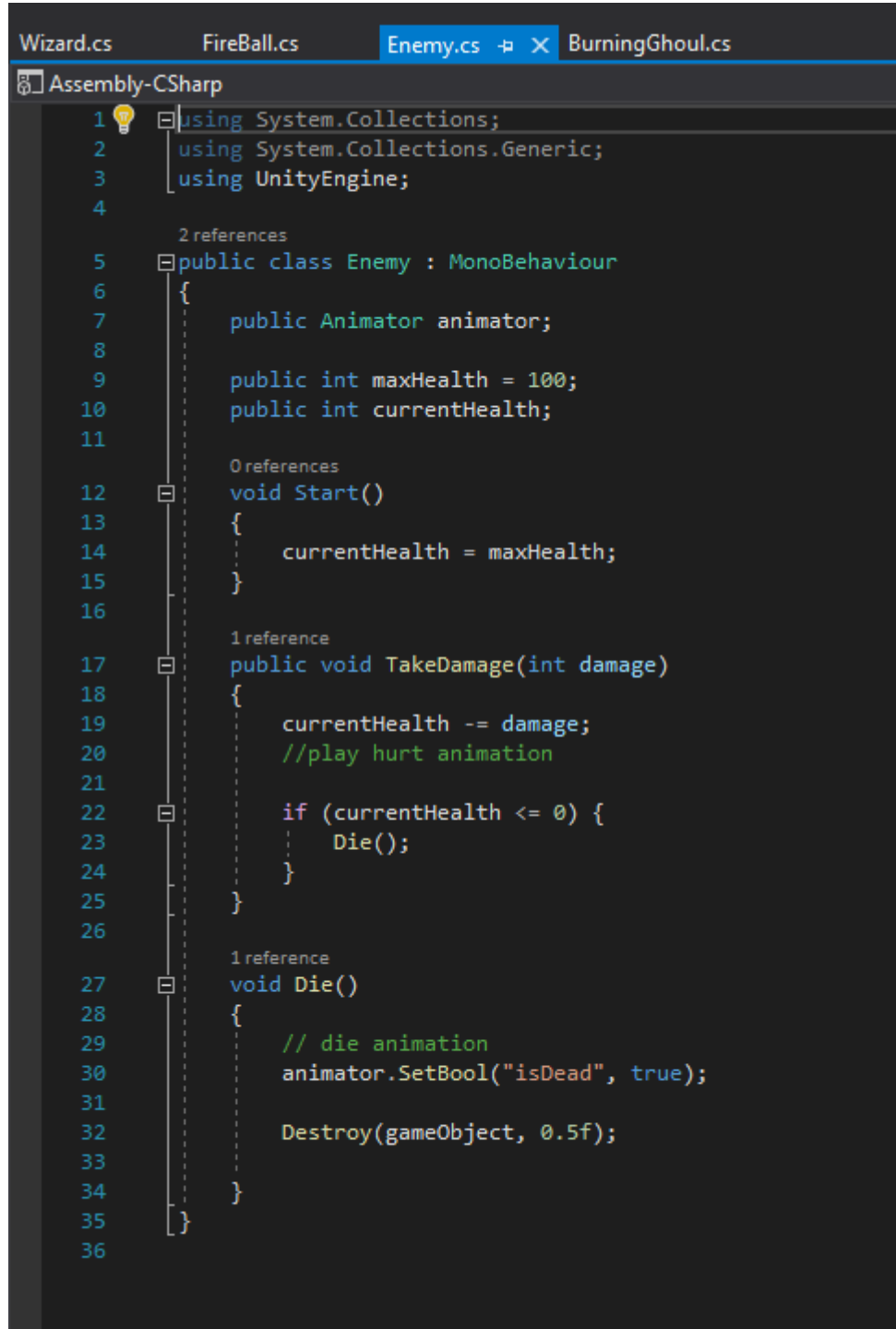
Oyuncunun ne kadar canı kaldığını göstermek için Canvas Objesi yaratarak buraya kalpler eklenmiştir.



Şekil.3.4.14.Oyuncunun Canını Gösteren UI , Oyun görünümü

Oyuncunun canı 100 dür. Şekil.3.4.14’de görüldüğü gibi oyun ekranında oyuncunun canı 10 tane kalple gösterilir. Can kaybettikçe kalpler kırmızıdan maviye döner.

### 3.5.DÜŞMAN KARAKTERLERİN YARATILMASI

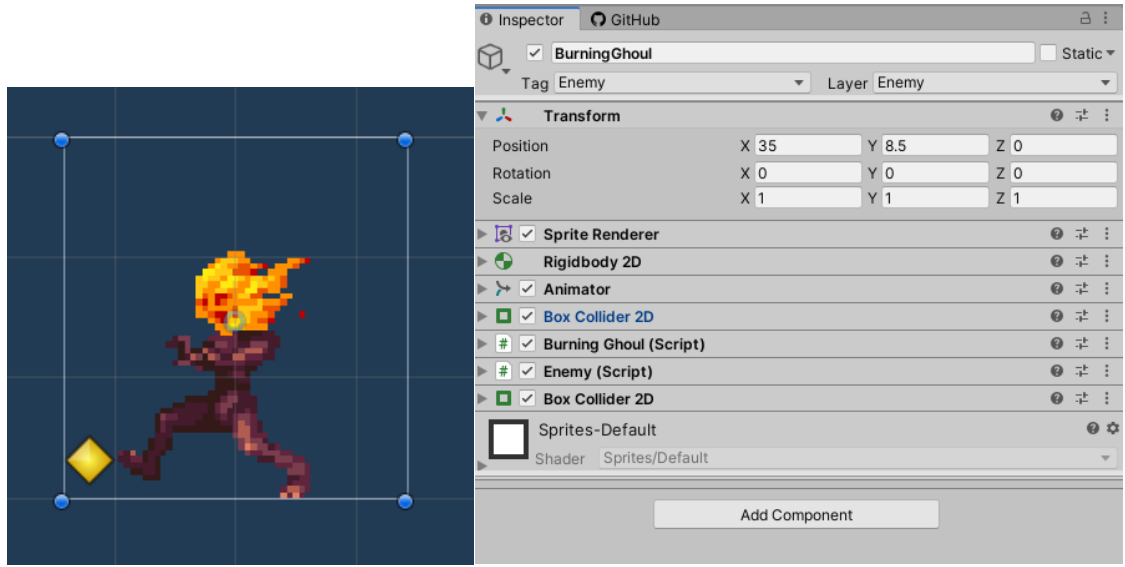


Şekil.3.5.1.Enemy Kodu

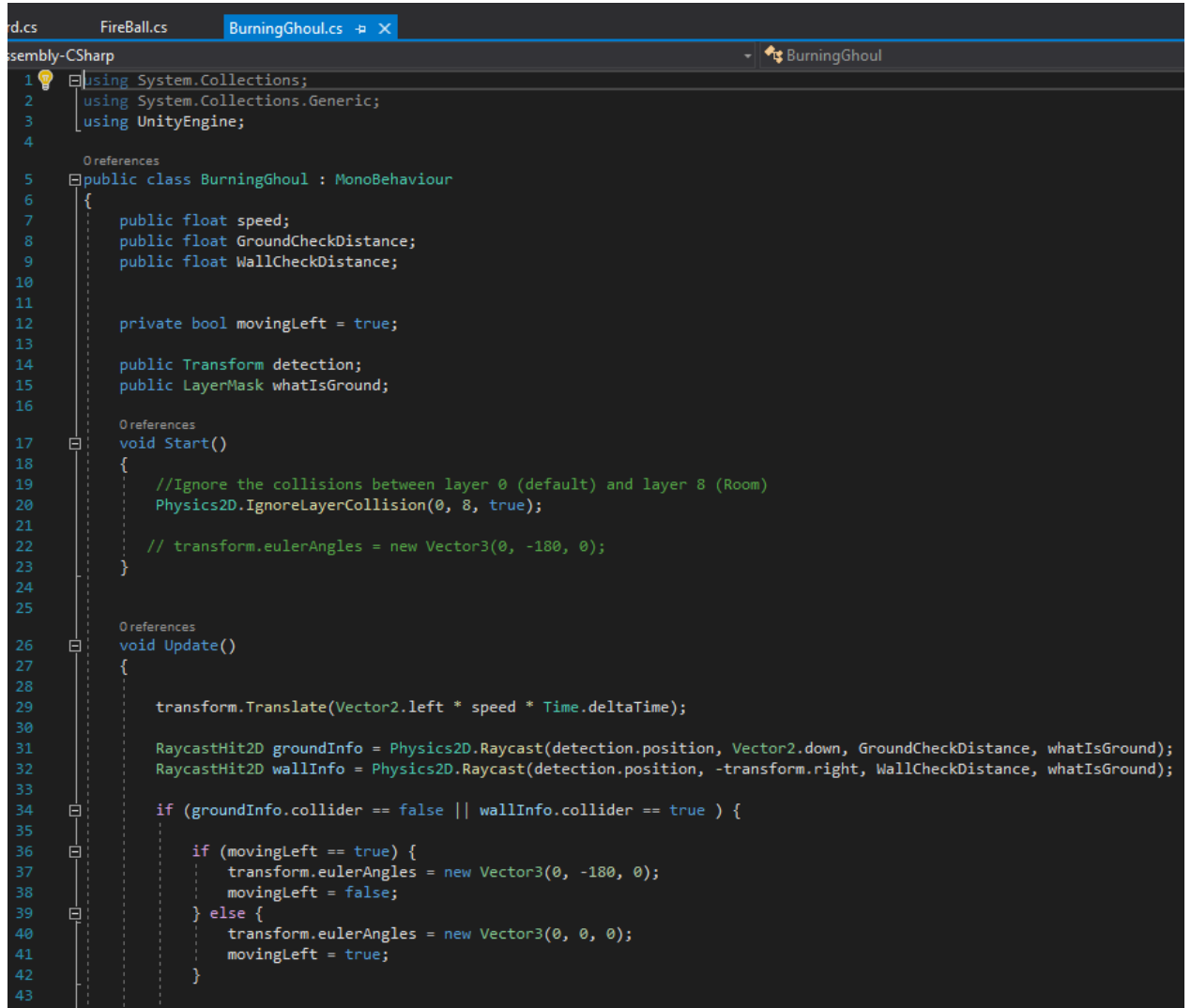
Bütün düşman karakterlerine Şekil.3.5.1’de görülen Enemy kodu eklenmiştir. Bu kod düşmanların oyuncu saldırdığı zaman zarar görmelerini ve ölmelerini sağlar. Sağlık puanları 0 ın altına düşünce ölürlr ve ölme animasyonu oynatıldıktan sonra bu kodun bağlı olduğu düşman karakteri sahneden silinir.

3 tane düşman karakteri vardır:

### 3.5.1. Burning Ghoul



Şekil.3.5.1.1.BurningGhoul



Şekil.3.5.1.2.BurningGhoul kodu

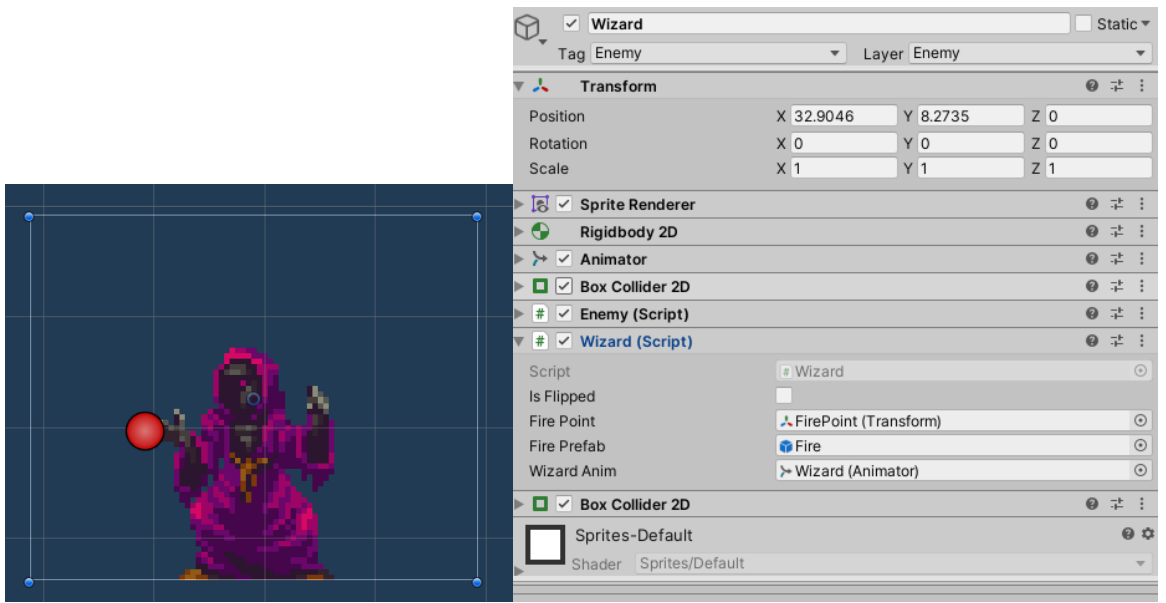


BurningGhoul karakteri bırakıldığı platformda önüne boşluk veya bir engel çıkmadıkça ileri ve geri ilerler. Bunuda karakterin önüne eklenmiş sarı noktadan aşağıya ve ileriye doğru ışın atarak orada bir önünde ne olduğunu belirleyerek yapar. Eğer önünde bir engel varsa veya önü platform sonuysa diğer tarafa dönüp o tarafa doğru koşar. Oyuncu ona çarparsa 10 sağlık puanı kaybeder.

```
50  
51     if (collision.CompareTag("Enemy")) {  
52         playerAnim.SetTrigger("hurt");  
53         TakeDamage(10);  
54     }  
55 }  
56
```

Şekil.3.5.1.2.BurningGhoul kodu devamı

### 3.5.2.Wizard



Şekil.3.5.2.1.Wizard

```
0 references  
5 public class Wizard : MonoBehaviour  
6 {  
7     Transform player;  
8  
9     public bool isFlipped = true;  
10  
11     //public float shootRate = 20f;  
12     float nextShoot = 0;  
13  
14     public Transform firePoint;  
15     public GameObject firePrefab;  
16  
17     public Animator WizardAnim;  
18  
19     0 references  
20     private void Start()  
21     {  
22         player = GameObject.FindGameObjectWithTag("Player").transform;  
23     }  
24
```

Şekil.3.5.2.2.Wizard kodu

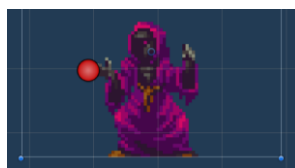
```

26 void Update()
27 {
28     FaceThePlayer();
29
30     if (Time.time > nextShoot) {
31         int randShootRate = Random.Range(5, 15);
32         nextShoot = Time.time + randShootRate;
33         Shoot();
34     }
35 }
36
37 }
38
39 1 reference
40 void FaceThePlayer()
41 {
42     Vector3 flipped = transform.localScale;
43     flipped.z *= -1f;
44
45     if (transform.position.x > player.position.x && isFlipped) {
46         transform.localScale = flipped;
47         transform.Rotate(0f, 180f, 0f);
48         isFlipped = false;
49     } else if (transform.position.x < player.position.x && !isFlipped) {
50         transform.localScale = flipped;
51         transform.Rotate(0f, 180f, 0f);
52         isFlipped = true;
53     }
54 }
55
56 1 reference
57 void Shoot()
58 {
59     WizardAnim.SetTrigger("shoot");
60
61     StartCoroutine(waiter());
62 }
63
64 1 reference
65 IEnumerator waiter()
66 {
67     //Wait for 0.7 seconds
68     yield return new WaitForSecondsRealtime(0.7f);
69
70     Instantiate(firePrefab, firePoint.position, firePoint.rotation);
71
72 }
73

```

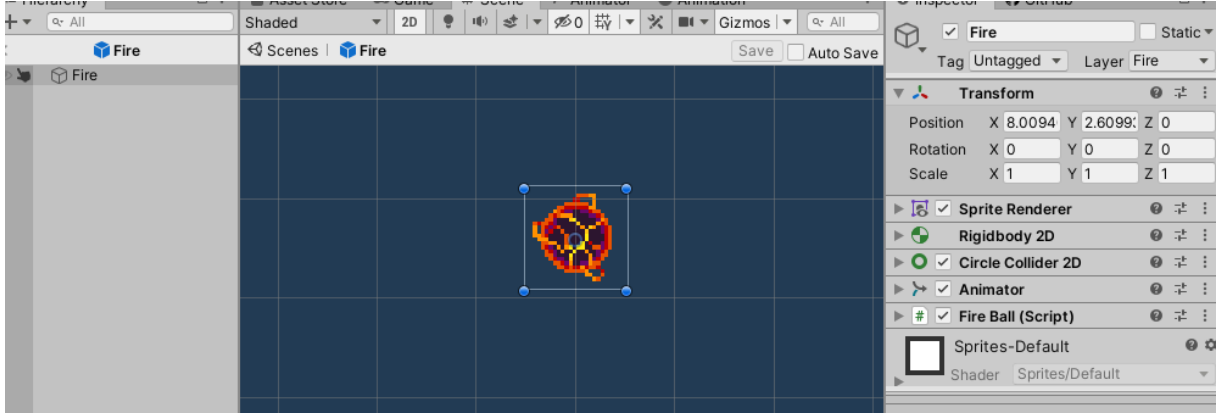
Şekil.3.5.2.3.Wizard kodu devamı

Wizard koduyla büyücü düşman karakteri her zaman oyuncunun bulunduğu yöne doğru bakar ve 5-15 sn arasında random bir zamanda düşmana ateş topu atar. Bunu yapmak için büyücü düşman karakterine FirePoint game objesi eklenmiştir. Bu nokta aşağıdaki kırmızı noktadır.

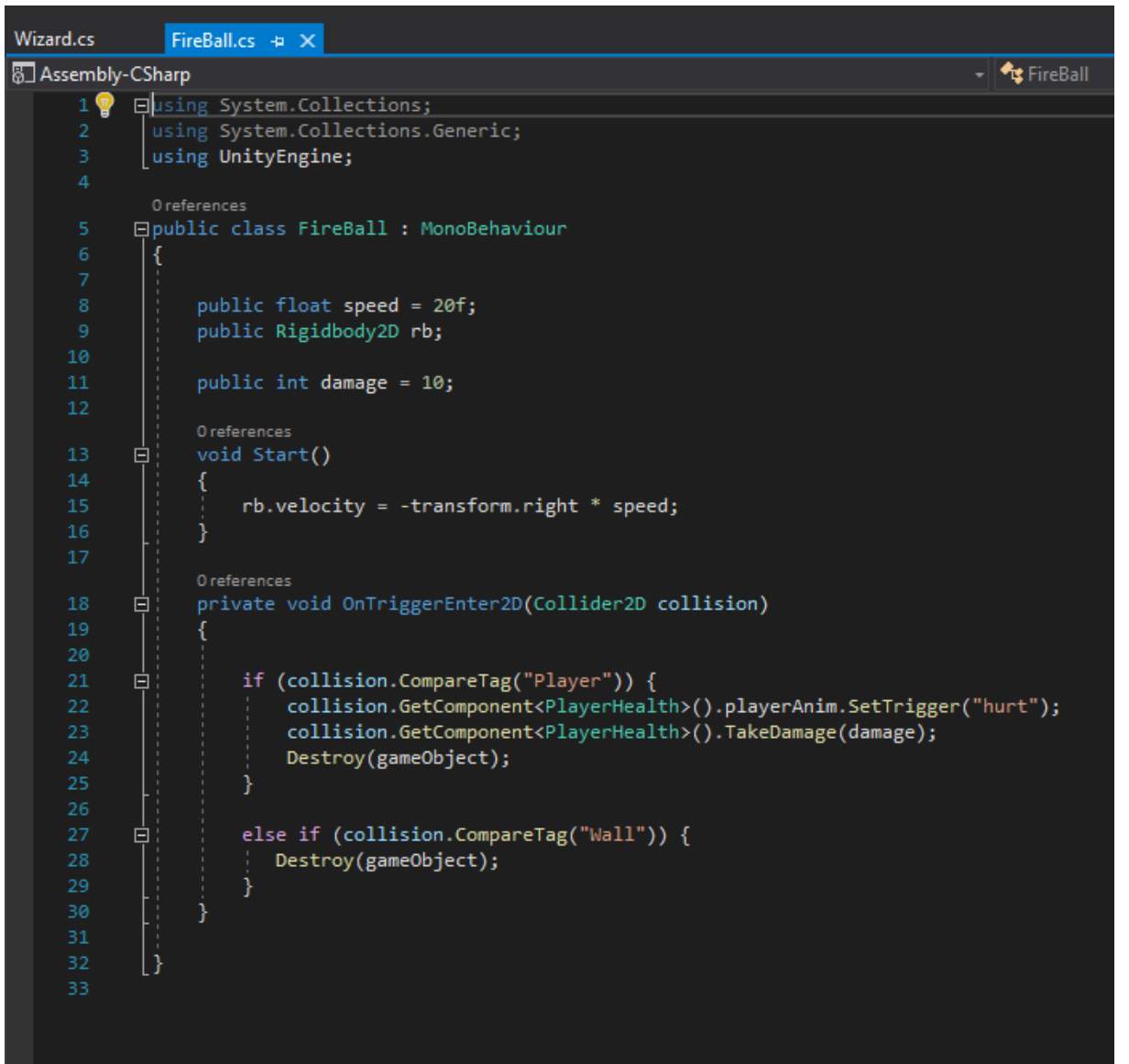


Şekil.3.5.2.4.Wizard FirePoint

Büyücü düşman karakteri ateş etmek istediği zaman bu kırmızı noktanın pozisyonunda Fireball game objesini oluşturur. Ateş topunun üzerinde bulunan FireBall kodu onu kontrol eder.



Şekil.3.5.2.5. FireBall

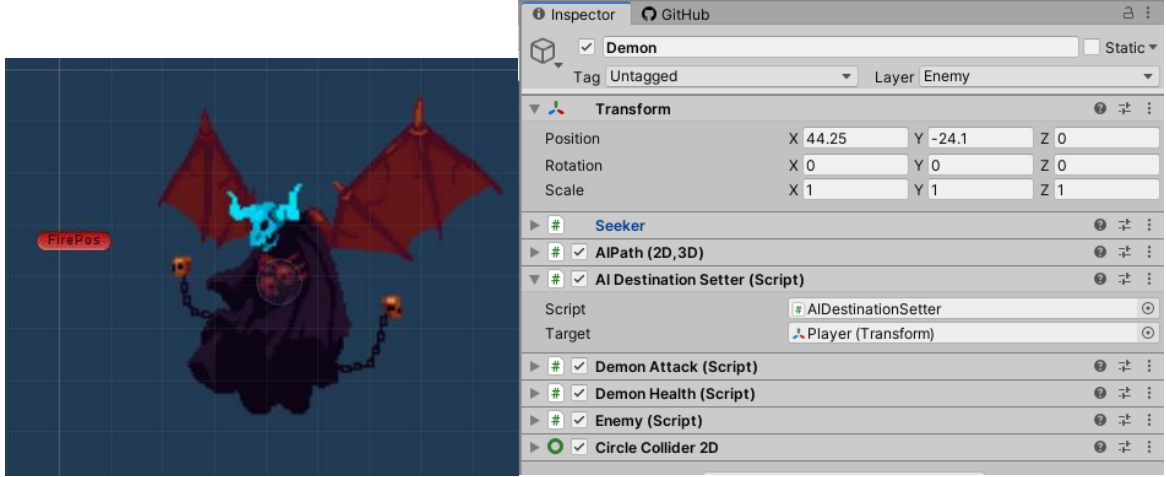


Şekil.3.5.2.5. FireBall kodu

FireBall kodu oluşturulan ateş topunun ileriye doğru hareket etmesini sağlar. Ateş topu oyuncu karakterine dokunduğu zaman oyuncunun PlayerHealth koduna oluşarak burdan yaralanma

animasyonunu oynatır ve oyuncuya 10 sağlık hasarı verir. Daha sonra bu kodun bağlı olduğu ateş topunu yok eder. Aynı zamanda eğer ateş topu düşmana çarpmadan duvara çarparsa yine yok edilir böylece oyunda gereksiz objelerin birikimi engellenir.

### 3.5.3. Demon



Şekil.3.5.3.1.Demon

4. bölümde oyunu zorlaştırmak için oyuna yeni bir düşman karakteri eklenir. Bu düşman karakteri diğer düşmanlara göre daha güçlüdür. Path finding kullanarak oyuncunun yerini bulur. Oyuncuya belirli bir mesafeye geldiği zaman ona ateş püskürür.

Oyuncu diğer düşman karakterini bir vuruşta öldürebilirken Demon karakterini öldürmek için 10 kere vurması gerekir çünkü bu karakterin canı 100'dür. Ekranda Demon karakterinin ne kadar canının kaldığını göstermek için aynı oyuncu karakterinde olduğu gibi bir Canvas objesi ve 10 tane kalp yaratılır. Demonun kalpleri başlangıçta mordur. Can kaybettikçe maviye dönerler.



Şekil.3.5.3.2.Demon Karakterinin Canının Ekranda Gösterilmesi

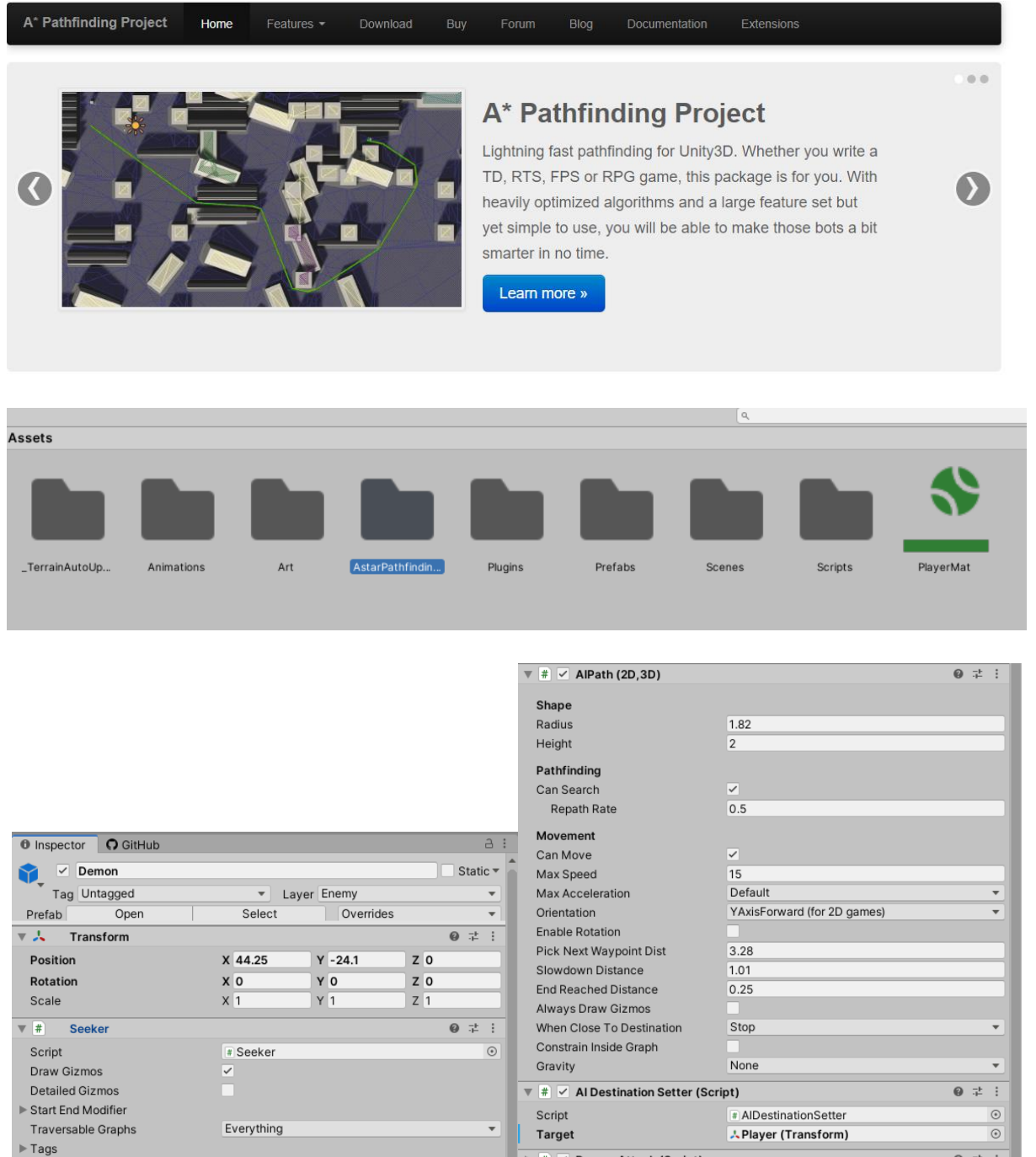
Düşmanın ne kadar canının kaldığını gösteren kalp sistemini aşağıdaki kod sağlar:



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class DemonHealth : MonoBehaviour
7  {
8      //public Animator animator;
9
10     public int maxHealth = 100;
11     //int currentHealth;
12     int health;
13
14     public int numberOfHearts = 10;
15
16     public Image[] hearts;
17     public Sprite fullHeart;
18     public Sprite emptyHeart;
19
20
21     void Start()
22     {
23         health = maxHealth;
24     }
25
26
27     private void Update()
28     {
29         health = GetComponent<Enemy>().currentHealth;
30         numberOfHearts = health / 40;
31
32         for (int i = 0; i < hearts.Length; i++) {
33
34             if (i < numberOfHearts) {
35                 hearts[i].sprite = fullHeart;
36             } else {
37                 hearts[i].sprite = emptyHeart;
38             }
39         }
40     }
41
42
43 }
44
```

Şekil.3.5.3.3.DemonHealth kodu

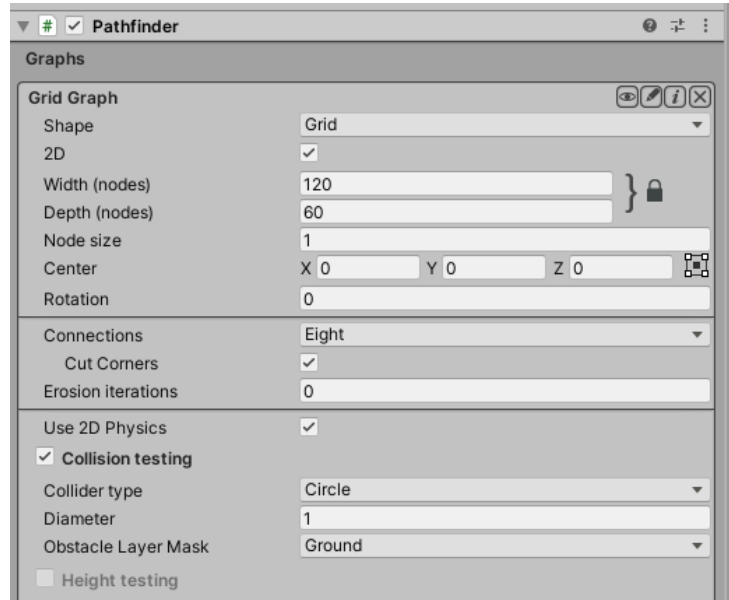
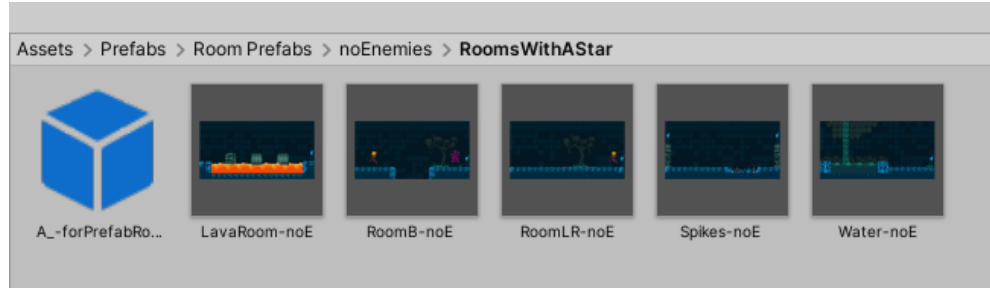
Demon düşman karakterine Path Finding eklemek için unity nin desteklediği A\* Path finding algoritma projesi , oyun projesine eklenmiştir.



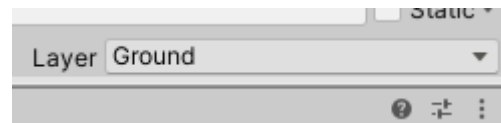
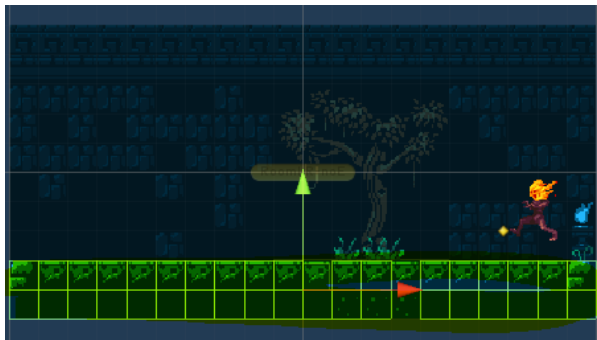
Şekil.3.5.3.4. A\* Path finding

Bu dosya projeye eklendikten sonra Demon düşman karakterine Seeker , Alpath ve AI Destination Setter objeleri eklenmiştir ve oyunun ihtiyaçlarına uygun şekilde düzenlenmiştir. Destination olarak oyunucunun konumu verilerek onu takip etmesi sağlanmıştır.

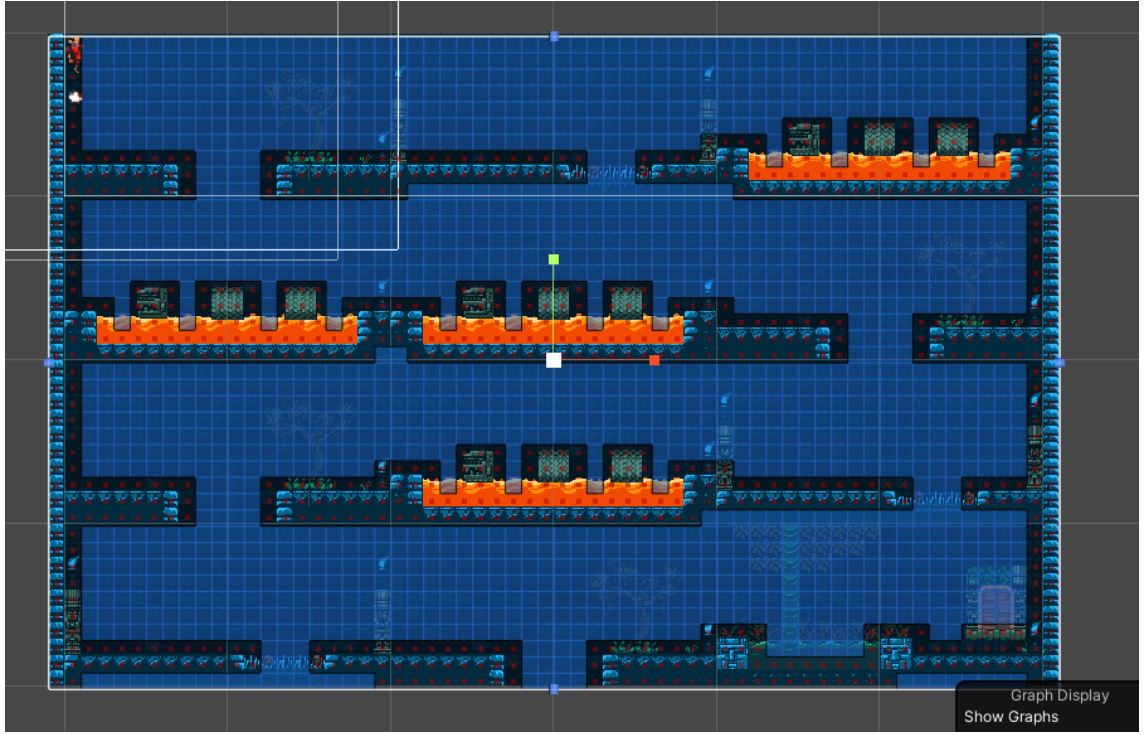
Düşman karakterin A\* pathfinding algoritmasını kullanarak oyuncuyu bulabilmesi için odalara A\* algoritması eklenerek yeniden düzenlenmesi gerekmiştir. Bu yüzden sadece bu bölümde kullanılmak üzere içinde A\* bileşenleri olan odalar tekrar oluşturulmuştur ve bunlar Level Generation objesine eklenerek bu bölüm oluşturulken A\* bileşenleri olan odaların kullanılması sağlanmıştır.



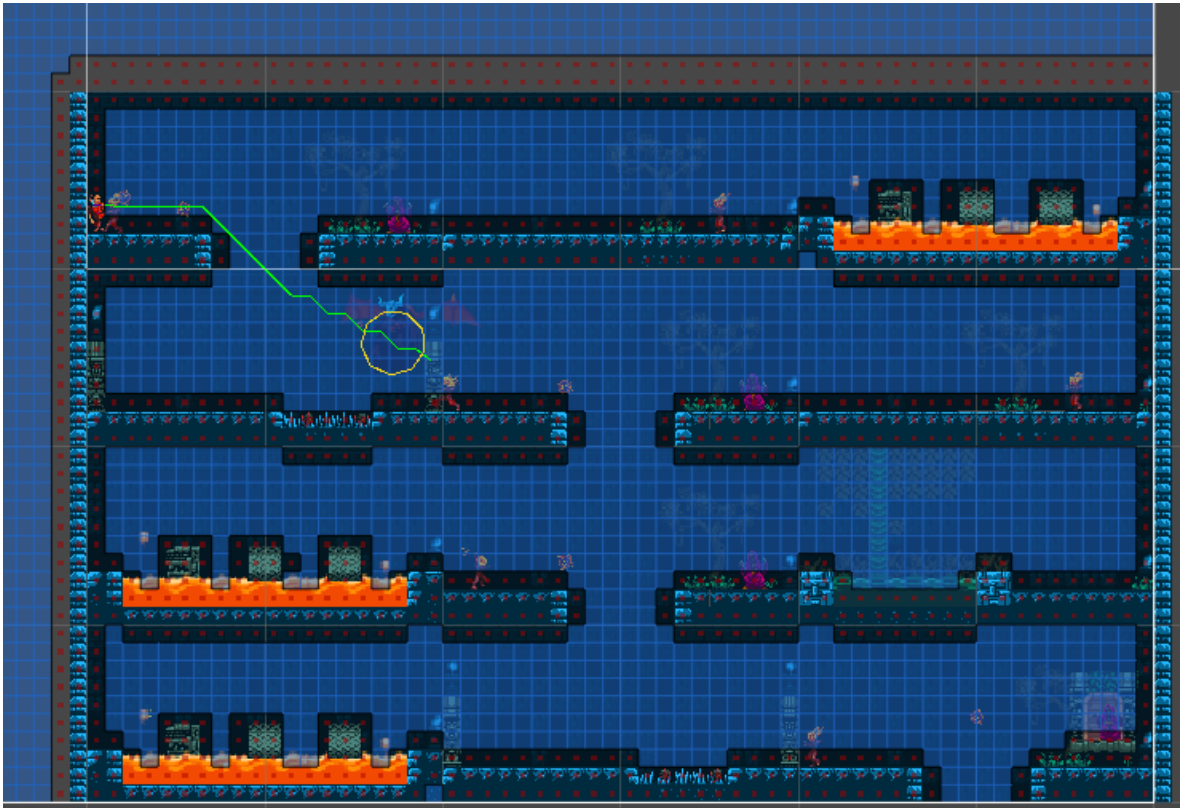
Odalara eklenen A\* objesi ile , PathfFinder bileşeni eklenir böylece oda gridlere ayrılır. Obstacle Layer Mask , yani düşmanın üzerinden geçemeyeceği bölümler belirlenir. Bu oyunda demon düşmanı için obstacle , Ground Layerıdır. bunun için her odacığa gidip , ground olana her yer için Ground Layerı seçilmiştir.



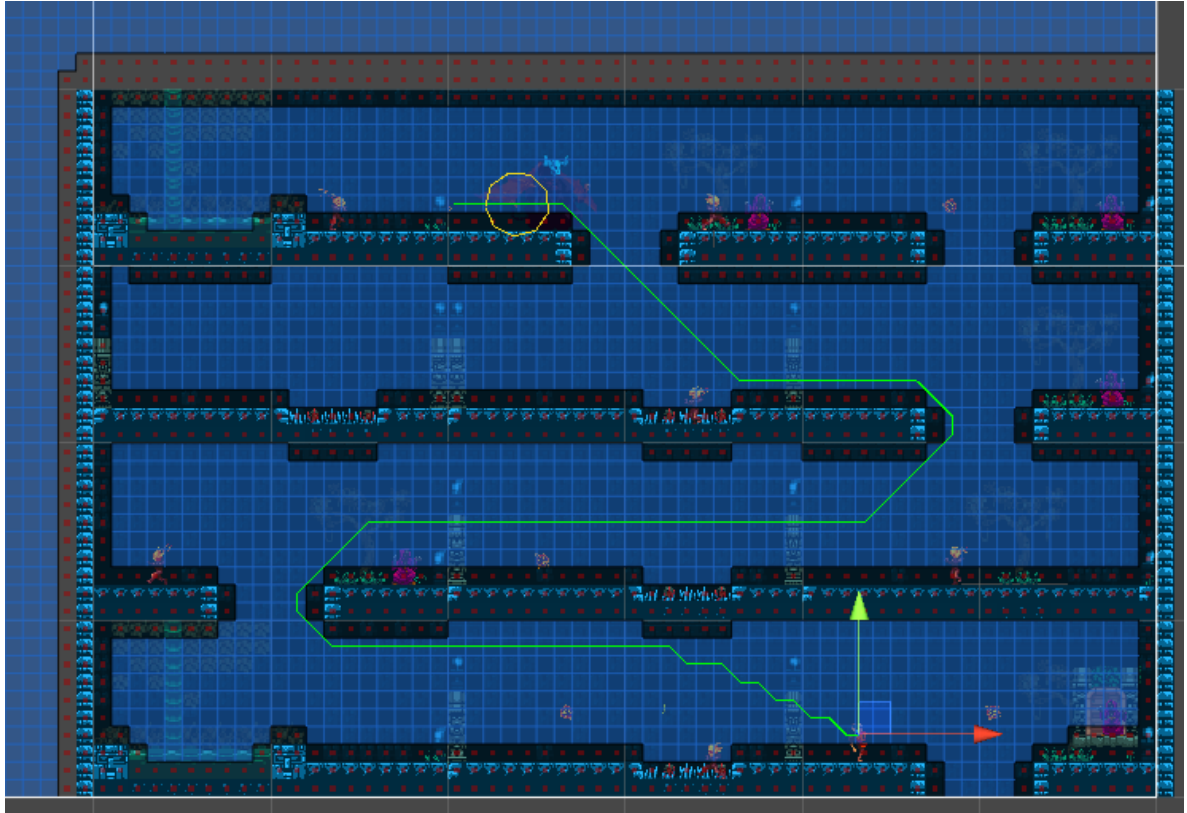




Level Genration yeni bölümü oluşturduğunda odalardaki ekli olan A\* algoritması ve path finder objesi birleşerek bütün bölüm üzerinde düşmanın üzerinden geçebileceği ve geçemeyeceği (obstacle ) yerlerin haritasını çıkarır. Yukarıda görülmüş mavi yerler Demon düşman karakterinin hareket edebileceği yerlerdir. Kırmızı noktalı yerler ise düşmanın hareket edemeyeceği yerlerdir. Oyunu başlattığımız zaman düşman karakterinin oyuncuya karşı bulduğu yolu scene penceresinde görebiliriz. Yeşil çizgiler düşmanın bulduğu yoldur.







```
DemonGFX.cs*  DemonFire.cs  DemonAttack.cs
Assembly-CSharp

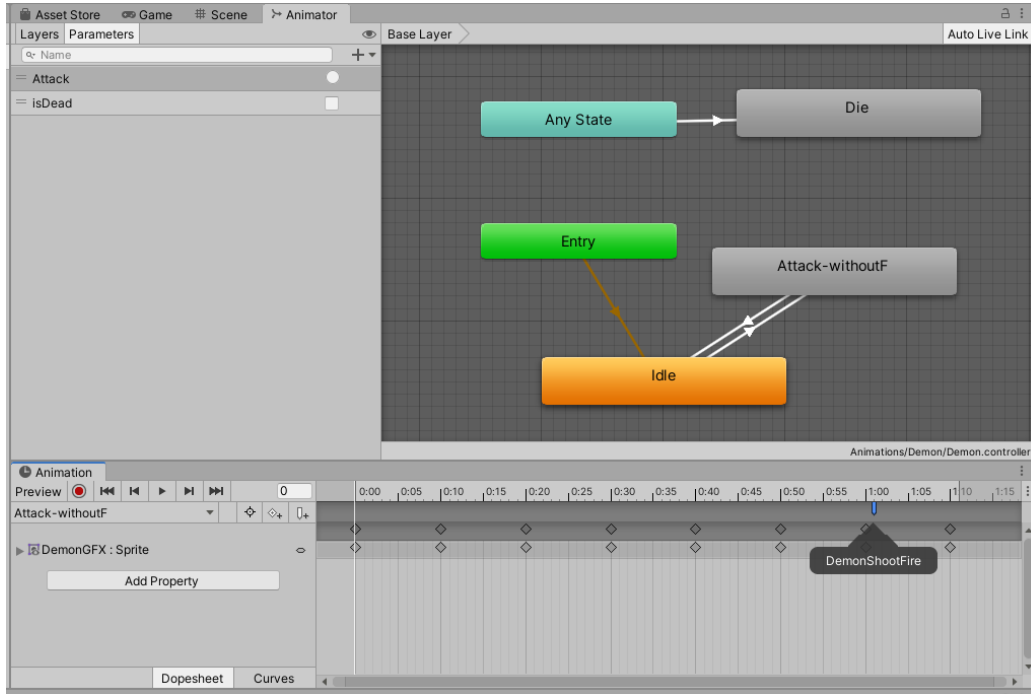
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Pathfinding;
5
6  public class DemonGFX : MonoBehaviour
7  {
8      public AIPath aiPath;
9      void Update()
10     {
11
12         if(aiPath.desiredVelocity.x >= 0.01f) {
13             transform.localScale = new Vector3(-1f, 1f, 1f);
14             //transform.Rotate(0f, 180f, 0f);
15         } else if(aiPath.desiredVelocity.x <= -0.01f) {
16             transform.localScale = new Vector3(1f, 1f, 1f);
17             //transform.Rotate(0f, 0f, 0f);
18         }
19     }
20 }
21
```

Demon düşman karakteri oyuncuya doğru hareket ederken hareket ettiği yöne bakması ve animasyonlarının yönünde doğru olması için yukardaki kod kullanılmıştır.

```
DemonGFX.cs*   DemonFire.cs   DemonAttack.cs  X
Assembly-CSharp  DemonAttack

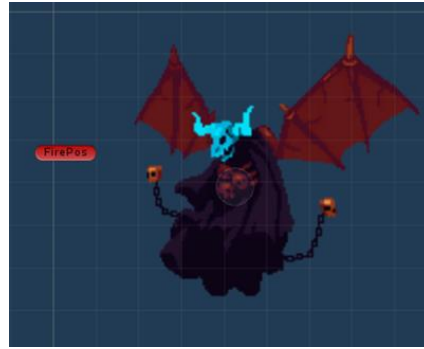
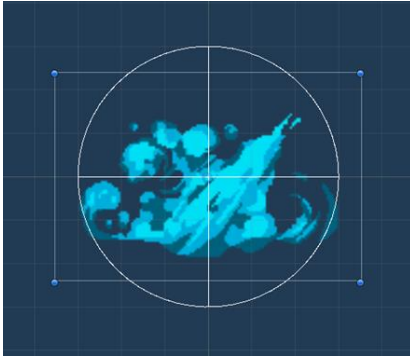
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  References
6  public class DemonAttack : MonoBehaviour
7  {
8      public Transform player;
9      public Rigidbody2D rb;
10     public Animator anim;
11
12     public float attackRange = 5f;
13     float nextShoot = 0;
14
15     public Transform firePos;
16     public GameObject demonFire;
17
18     References
19     void Update()
20     {
21         if (Vector2.Distance(player.position, rb.position) <= attackRange && (Time.time > nextShoot)) {
22             int randShootRate = Random.Range(5, 7);
23             nextShoot = Time.time + randShootRate;
24             anim.SetTrigger("Attack");
25             //DemonFire();
26         }
27     }
28 }
```

Demon düşman karakterinin oyuncu belirli bir mesafeye girdiği zaman oyuncuya saldırması için yukardaki kod kullanılmıştır. Düşman oyuncuya yaklaştığı zaman Animator'a bir trigger yollar ve Attack animasyonu oynar. Bu animasyon üzerinden istediğimiz noktada DemonShootFire fonksiyonu çağrılır.



DemonShootFire fonksiyonu çağrıldığı zaman demon karakterinin önünde bulunan firePos objesinin yerinde demonFire oluşturur.

```
References
public void DemonShootFire() // this function would be called from animator state machine when demon
{
    if (rb.transform.localScale.x < 0) { // looking left
        Instantiate(demonFire, firePos.position, firePos.rotation * Quaternion.Euler(0f, 180f, 0f));
    } else if (rb.transform.localScale.x > 0) { // looking right
        Instantiate(demonFire, firePos.position, firePos.rotation);
    }
}
```



```

s* DemonFire.cs DemonAttack.cs
CSharp DemonFire
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

References
public class DemonFire : MonoBehaviour
{
    public Rigidbody2D rb;
    public int damage = 10;
    // GameObject player;

    public float attackRange = 3f;
    public LayerMask attackMask;

    /*
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Debug.Log("demonfire collide with player");
        if (collision.CompareTag("Player")) {
            collision.GetComponent<PlayerHealth>().playerAnim.SetTrigger("hurt");
            collision.GetComponent<PlayerHealth>().TakeDamage(damage);
            Destroy(gameObject);
        }
    }
    */

    References
    public void Hurt()
    {
        Collider2D colInfo = Physics2D.OverlapCircle(rb.transform.position, attackRange, attackMask);
        if(colInfo != null) { // collide with player
            colInfo.GetComponent<PlayerHealth>().playerAnim.SetTrigger("hurt");
            colInfo.GetComponent<PlayerHealth>().TakeDamage(damage);
        }
    }

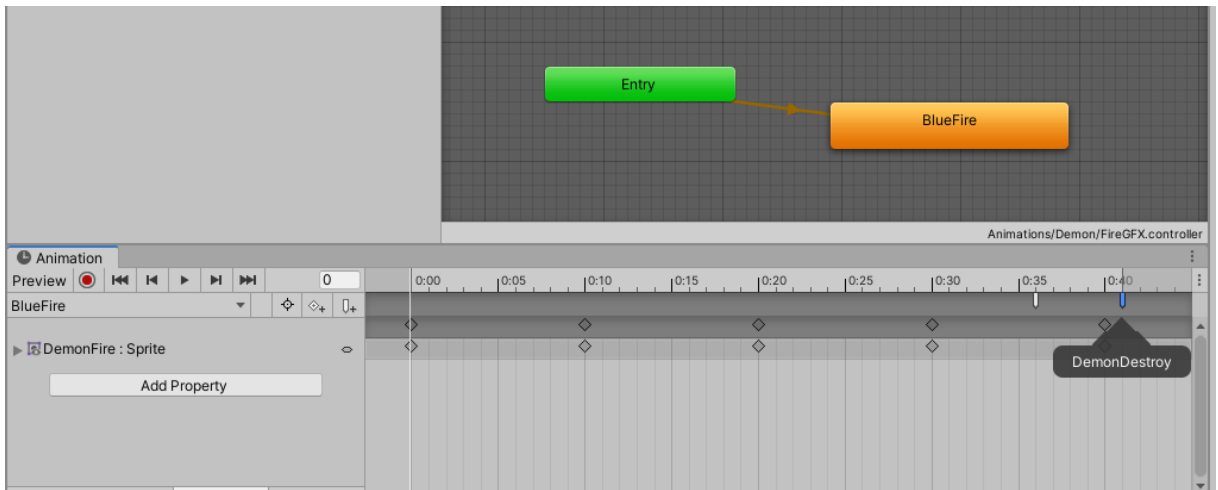
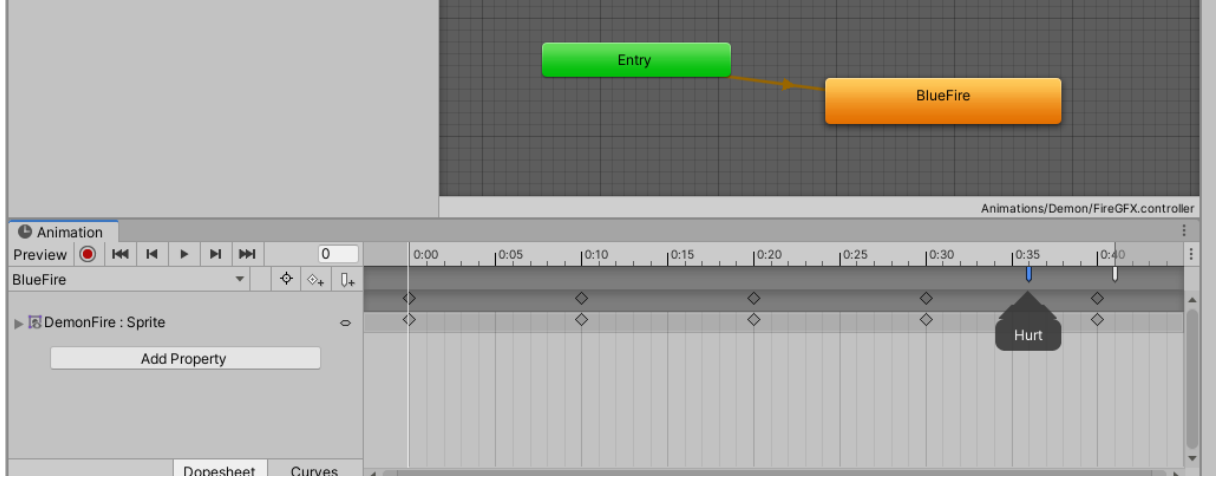
    References
    public void DemonDestroy()
    {
        Destroy(gameObject);
    }

    // to draw the attack range so we can change it as we like
    References
    private void OnDrawGizmosSelected()
    {
        if (rb.transform.position == null)
            return;

        Gizmos.DrawWireSphere(rb.transform.position, attackRange);
    }
}

```

DemonFire kodu demonFire objesine eklenmiştir. Ve bu kod demonun püskürttüğü ateşin oyuncuya zarar vermesini sağlar ve işimizin bittiği zaman demonFire objesinin sahneden silinmesini sağlar. Hurt ve DemonDestroy fonksiyonları Animator üzerinden animasyon üzerinde istediğimiz noktadan çağrılır.



```
27 | 0 references  
28 | public void Hurt()  
29 | {  
30 |     Collider2D colInfo = Physics2D.OverlapCircle(rb.transform.position, attackRange, attackMask);  
31 |     if(colInfo != null) { // collide with player  
32 |         colInfo.GetComponent<PlayerHealth>().playerAnim.SetTrigger("hurt");  
33 |         colInfo.GetComponent<PlayerHealth>().TakeDamage(damage);  
34 |     }  
35 | }  
  
36 | 0 references  
37 | public void DemonDestroy()  
38 | {  
39 |     Destroy(gameObject);  
40 | }
```

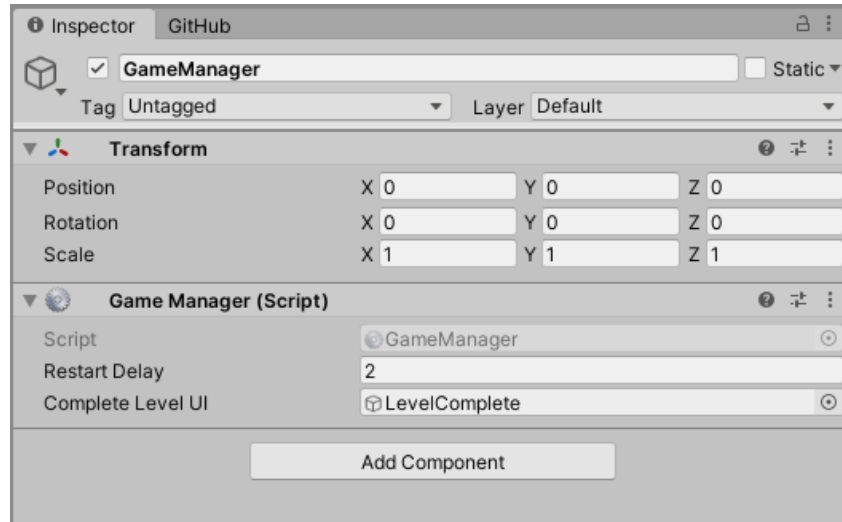
### 3.6. KULLANICI ARAYÜZÜ VE LEVELLERİN OLUŞTURULMASI

Oyunda 7 sahne bulunmaktadır. Bunların oyundaki sırası Şekil.3.6.1 de görülebilir.

Oyundaki sahneler GameManager oyun objesi ile kontrol edilir. Şekil.3.6.2 de görüleceği üzere GameManager objesine LevelComplete oyun objesi eklenmiştir. Level Complete objesi bir Canvas görüntü objesi olup kullanıcı arayüzü için tasarlanmıştır. Bu obje Şekil.3.6.3 te görülür. Oyuncu bir bölümü (level) yendiği zaman ilk önce bu ekran görülür. Daha sonra yeni bölüm yüklenir.



Şekil.3.6.1.Oyundaki sahneler

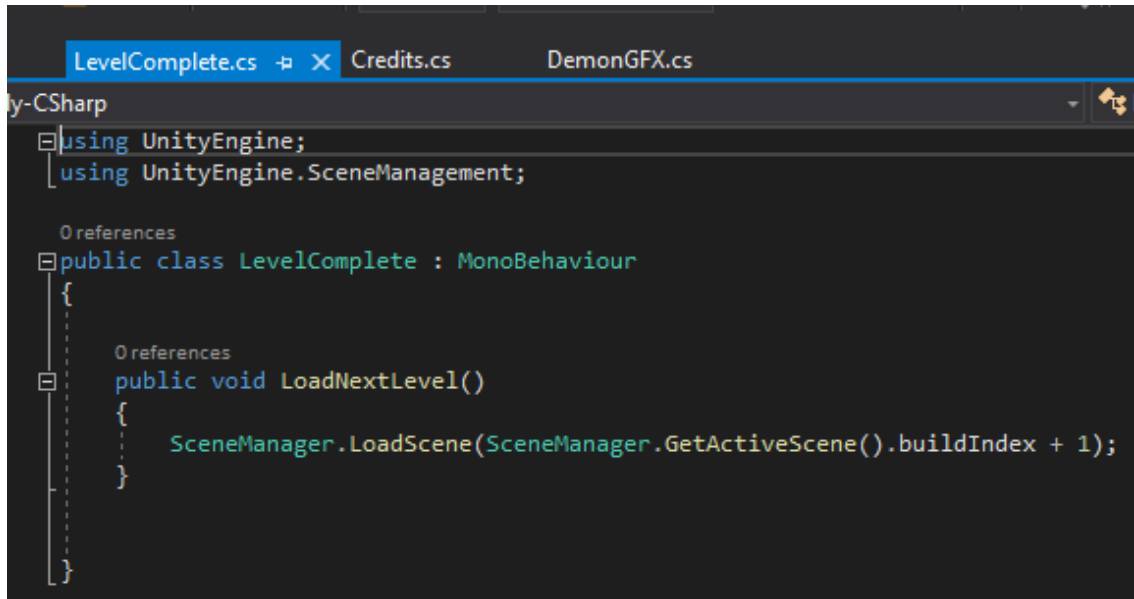


Şekil.3.6.2.GameManeger objesi



Şekil.3.6.3.LevelComplete objesi

Oyunda sahneler arasında geçiş yapmak için LevelComplete kodu kullanılır . Bu kod Şekil.3.6.4’ te görülebilir. Bu kodda bulunan LoadNextLevel() fonksiyonu yeni bölüm yükelenmek istenildiğinde çağrılır.

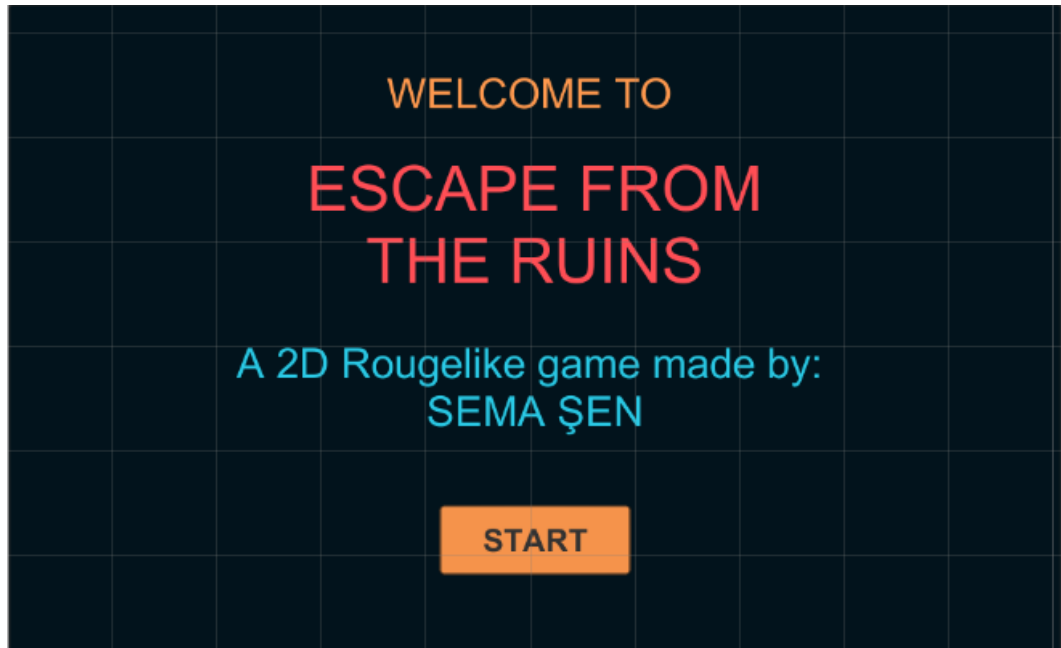


```
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelComplete : MonoBehaviour
{
    public void LoadNextLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}
```

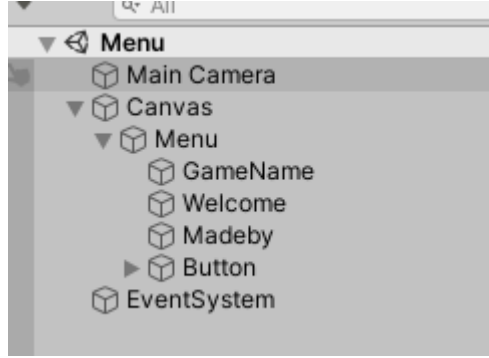
Şekil.3.6.4.LevelComplete kodu

Oyun başladığında ilk görülen sahne Şekil.3.6.5’te görülen Menu sahnesidir.



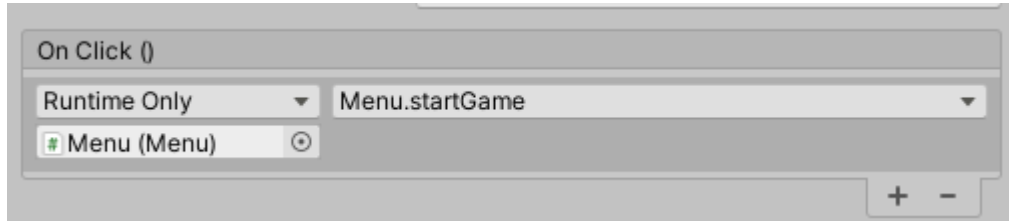
Şekil.3.6.5.Menu sahnesi

Menu sahnesi kullanıcı arayüzü sayfası olup Şekil.3.6.5’ te görüleceği üzere Canvas objesi ve buna bağlı üç text objesi ile bir Button objesinden oluşur.



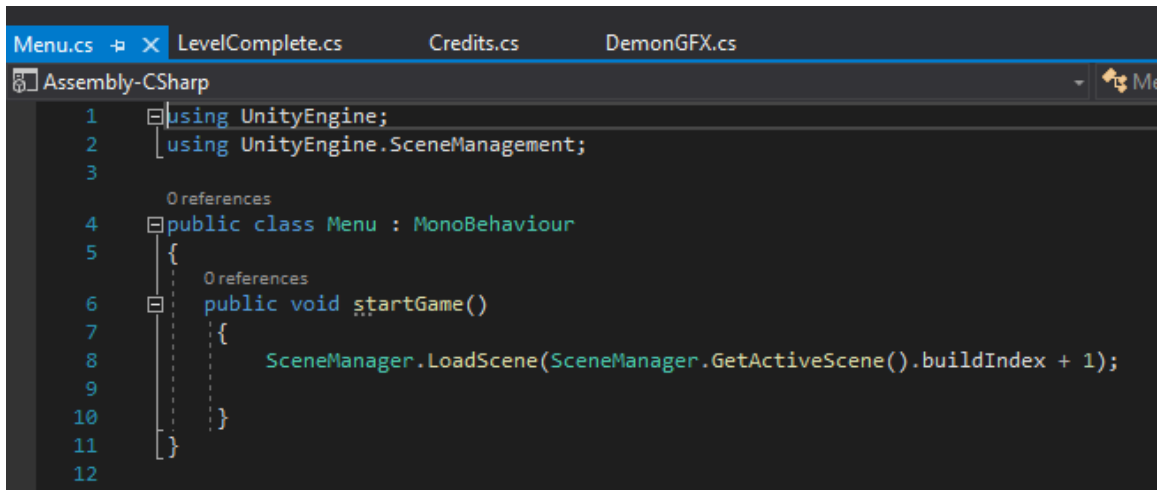
Şekil.3.6.5.Menu sahnesi bileşenleri

Şekil.3.6.6’ da görüleceği üzere Button objesi üzerine On Click() metotunda Menu kodundan çağrılan startGame fonksiyonu eklenmiştir.



Şekil.3.6.6.Button On Click() metottu

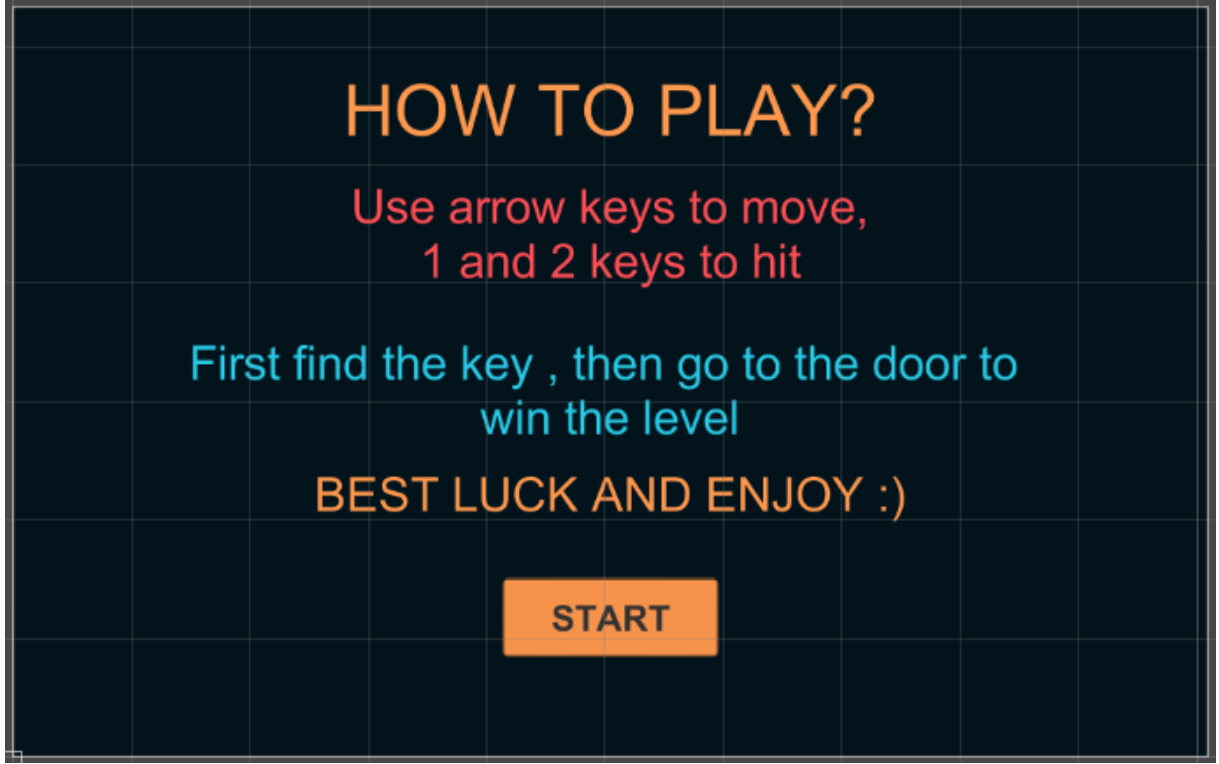
Şekil.3.6.7’ de görüleceği üzere Menu kodundaki startGame() fonksiyonu içinde bulunduğu sahneden sonraki sahneyi yükler.



Şekil.3.6.7.Menu kodu



Menu sahnesinden sonra Şekil.3.6.8’ de görülen Rules sahnesi yüklenir. Bu sahnede Menu sahnesinde olduğu gibi Canvas objesi üzerinde dört text objesi ve yine bir Button objesi vardır. Bu buttuna da aynı Menu sahnesinde olduğu gibi On Click() metotunda Menu kodundan çağrılan startGame fonksiyonu eklenmiştir. Ve yine aynı şekilde Menu kodundaki startGame() fonksiyonu içinde bulunduğu sahneden sonraki sahneyi yükler.



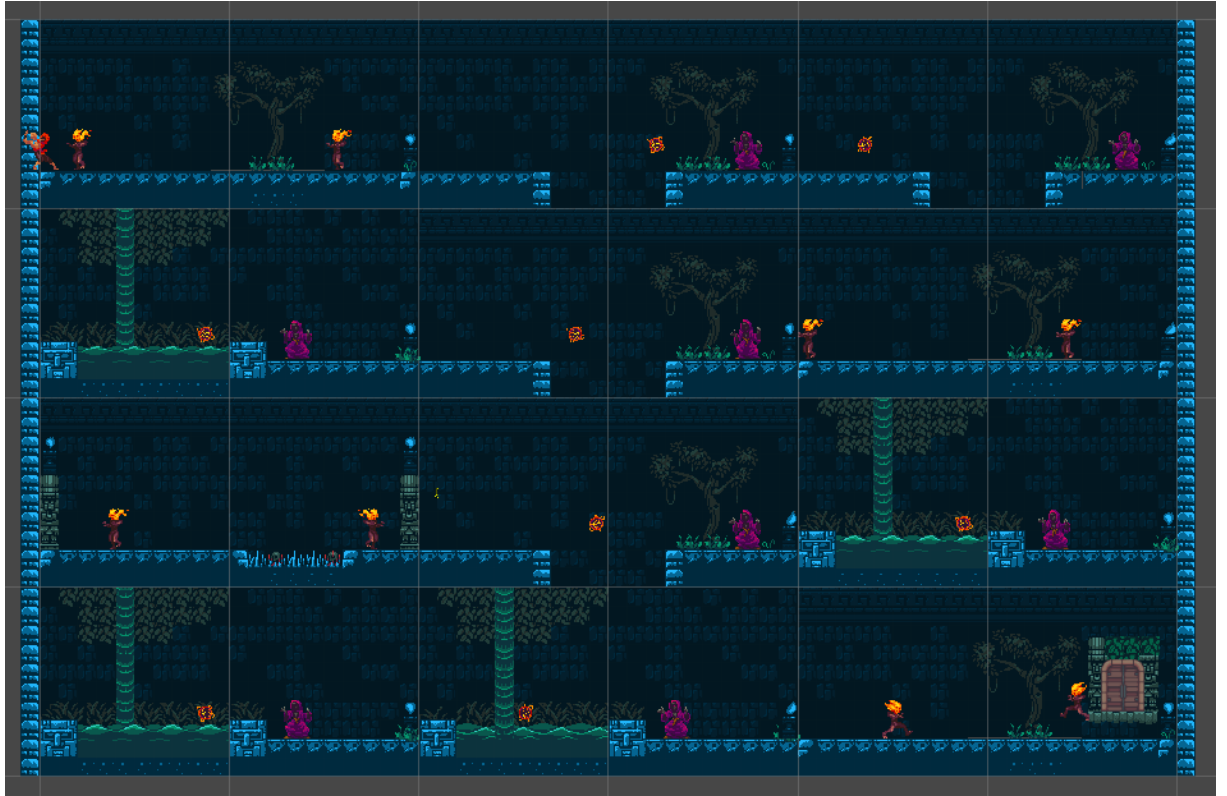
Şekil.3.6.8.Rules Sahnesi

Daha sonra sırasıyla birinin ardından Scene1 , Scene2 ve Scene3 (Şekil.3.6.9) sahneleri yüklenir. Bu sahneler birbirlerinin aynısı olup sahne başlangıcında ilk önce LevelGeneration rastgele bir labirent oluşturur. Bu sahnelerde Demon düşman karakteri bulunmaz , sadece BurningGhoul ve Wizard düşman karakterleri bulunur.

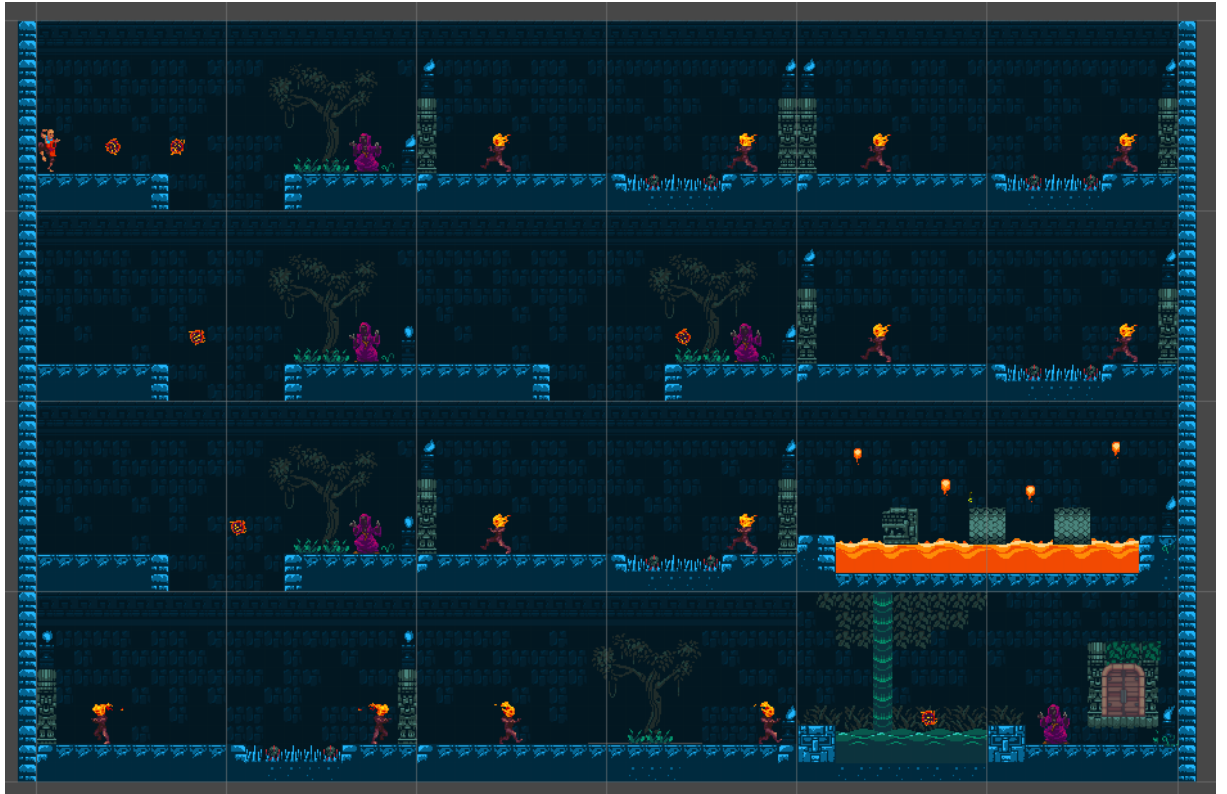


Şekil.3.6.9. Birinci , İkinci ve Üçüncü Sahneler

Oyunun bu sahnelerinden örnek resimler ( Şekil.3.6.10 , Şekil.3.6.11 )

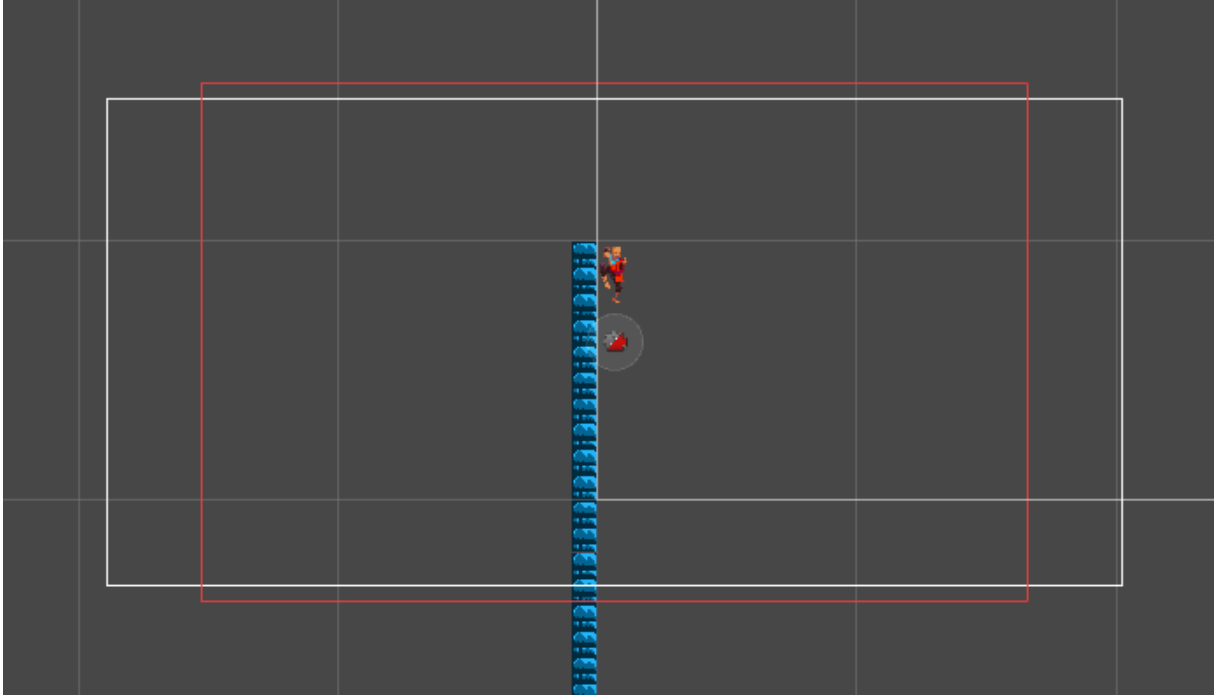


Şekil.3.6.10. Örnek 1



Şekil.3.6.11. Örnek 2

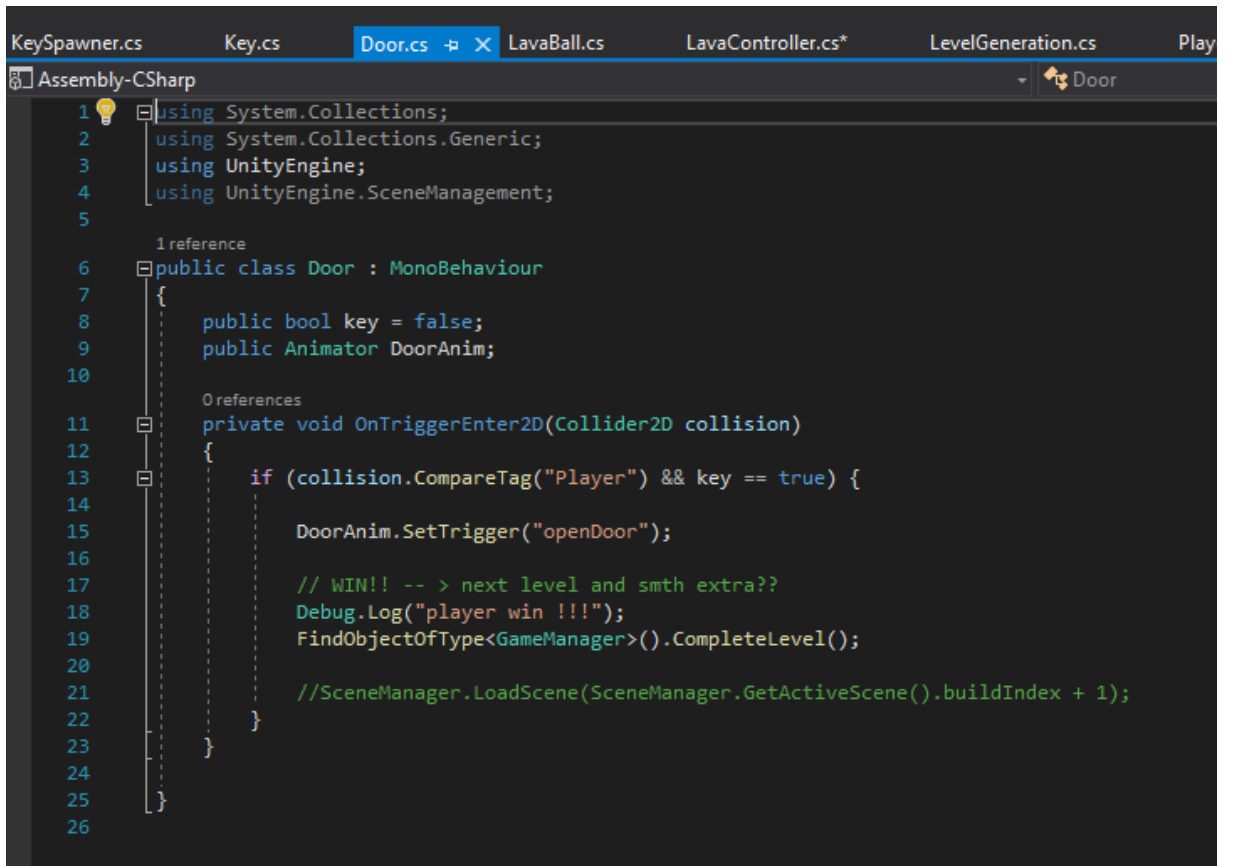
Şekil.3.6.12’ de görüleceği üzere kamera oyuncu karakterine odaklanmıştır bu yüzden oyun oynanırken Şekil.3.6.13’dedi gibi gözüktür , ekranda bütün labirent gözükmeyen sadece oyuncunun bulunduğu yerler gözüktür ve oyuncu hareket ettikçe kamera oyuncuyu takip eder.



Şekil.3.6.12. Camera View



Şekil.3.6.13. Oyunun Oyun Modunda Görünüşü



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.SceneManagement;
5
6 public class Door : MonoBehaviour
7 {
8     public bool key = false;
9     public Animator DoorAnim;
10
11     private void OnTriggerEnter2D(Collider2D collision)
12     {
13         if (collision.CompareTag("Player") && key == true) {
14
15             DoorAnim.SetTrigger("openDoor");
16
17             // WIN!! -- > next level and smth extra??
18             Debug.Log("player win !!!");
19             FindObjectOfType<GameManager>().CompleteLevel();
20
21             //SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
22         }
23     }
24 }
25
26
```

Şekil.3.6.14.Door kodu

Bölümler arasında geçiş Şekil.3.6.14'deki Door koduyla olur. Door kodunda key adlı bir bool değişkeni vardır. Bu değişken başlangıçta false dur. Oyuncu anahtarı bulduğu zaman bu değişken true olur.Eğer oyuncu anahtarı bulmadan kapıya gelirse kapı açılmaz. Oyuncu anahtarı bulur ve kapıya gelirse kapı açılır. ( kapı açılma animasyonu oynatılır.) Daha sonra GameManager kodundaki CompleteLevel fonksiyonu çağrılır. Buda bu bölümü bitirir ve yeni sahneyi yükler

Level 1 'den sonra Level 2 yüklenir.

Level 2 'den sonra Level 3 yüklenir.

Level 3 'ten sonra Level 4 yüklenir.

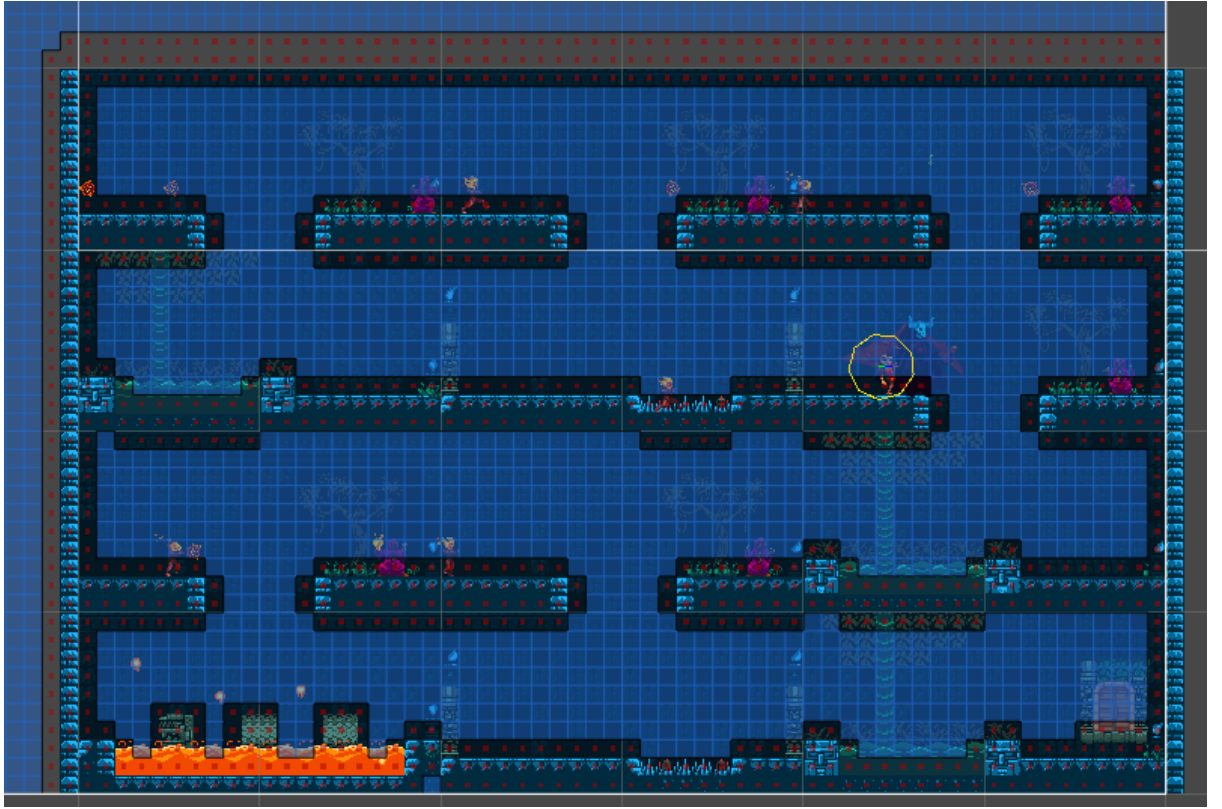
Level 4 sahnesi level 1 , level 2 , level 3 bölümlerinden farklıdır. Şekil.3.6.15’ teki oyun görünümde görüleceği üzeri bu bölümde Demon düşman karakteri vardır. Aynı zamanda Demon karakterinin ne kadar canı kaldığını gösteren kalpler şeklinde UI elementleri vardır. (Bunların işleyişi bölüm **3.5.3. Demon** ‘da detaylı bir şekilde anlatılmıştır. )



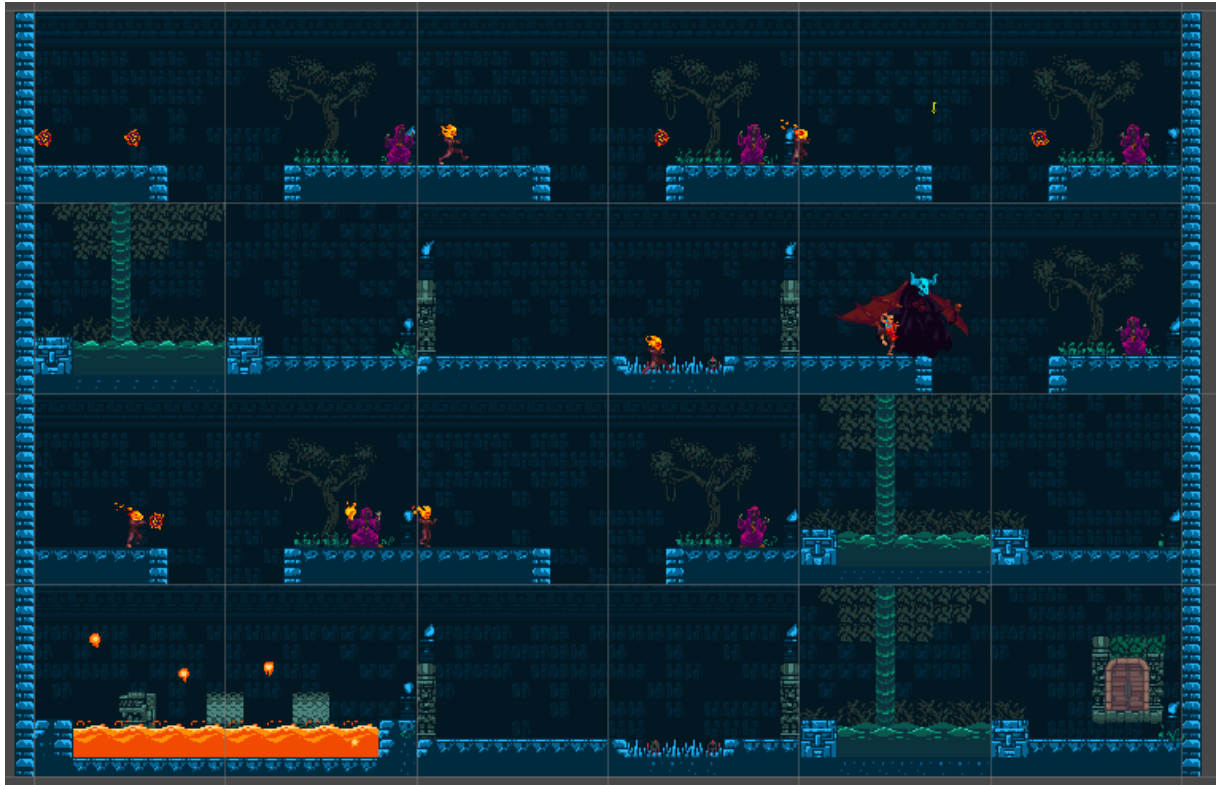
Şekil.3.6.15.Level4 oyun görünümü

Level4 de diğer bölümlerde olduğu gibi LevelGenration’nın labirenti oluşturmasıyla başlar ama bu bölümde kullanılan odacıkların içerisinde A\* Path finding bileşenleri vardır. Bu yüzden odacıklar birleşince Demon karakterinin üzerinden geçebileceği ve geçemeyeceği yerlerin haritasını çıkarır. Şekil.3.6.16 ( bir sonraki sayfa ) da görüldüğü üzere Mavi kısımlar düşmanın ilerleyebileceği yerlerdir . Kırmızı noktalı kısımler ise düşmanın üzerinden geçemeyeceği kısımlerdir. (Bunların işleyişi bölüm **3.5.3. Demon** ‘da detaylı bir şekilde anlatılmıştır. )

Şekil.3.6.17 de bu bölümde oluşturulan rastgele bir labirenti görebiliriz.



Şekil.3.6.16.Demon Hareket Haritası

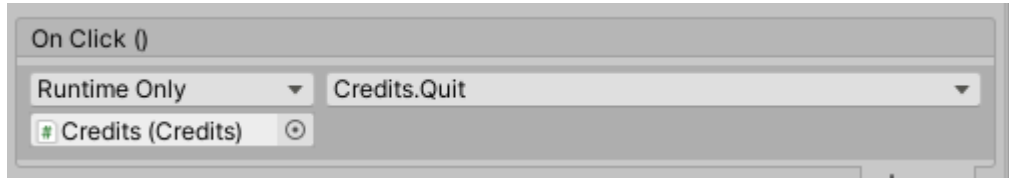


Şekil.3.6.17.Level4 örnek

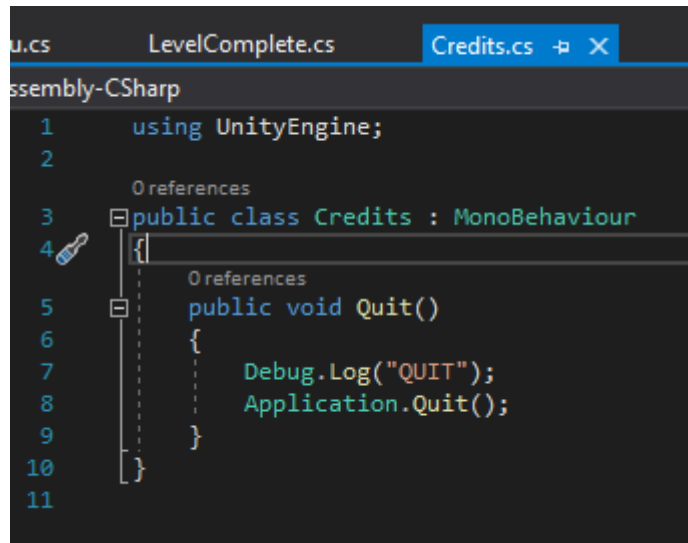
Level4 ‘ten sonra Şekil.3.6.18’de görülen Credits sahnesi yüklenir.Bu sahnede Canvas objesi üzerinde üç tane text objesi ve bir button objesi bulunmaktadır. Şekil.3.6.19’da bu button üzerinde buluna on click fonksiyonu görülür. Bu fonksiyon buttona tıklandığı zaman Şekil.3.6.20’deki Credits kodundan Quit fonksiyonunu çağırır.Quit fonksiyonu oyunu kapatır.



Şekil.3.6.18.Credits



Şekil.3.6.19.Credits Button Onclick fonksiyonu



Şekil.3.6.20.Credits Kodu



#### 4.TARTIŞMA VE SONUÇ

Projenin bitiminde ortaya çıkan oyunda level dizaynında procedural generation kullanılarak her seferinde farklı leveler üretilmiştir. Oyuncu olmayan karakterlerin davranışlarında yapay zeka kullanarak düşmanlara akıllı davranışlar kazandırılmıştır. Tüm bunlar oyuncu için daha zevkli bir oyun deneyimi yaratmıştır.

Roguelike türü yeni bir oyun türü olamamasına ragmen çok bilinen bir tür değildir. Ancak roguelike oyun türü oynaması da geliştirmesinde zor ve eğlenceli bir olduğu için roguelike türü oyunlar geliştirilmeye devam etmekte her geçen gün yeni bir oyun bu türe eklenmektedir.

Roguelike türü yapısı gereği yenilikçi bir türdür. Bu türün tanımında bulunan kurallar (permadeath , procedural level design , tilebased olmak ve turn based olmak gibi ) oyunun geliştiricisini yaratıcı olmaya iter.

Oyunun diğer türlere göre zor olaması , oyuna yeniden başlandığında farklı bir level dizaynıyla karşılaşılması , düşmanların akıllı davranışları bu türü zevkli ve yeniden oynalanabilir hale getirir.

Bu projede oynaması zevkli , roguelike türünün özelliklerine uygun 2 boyutlu bir oyun geliştirilmiştir. Bu oyunla ve yıl sonunda yapılan sunumla birlikte roguelike türü ve oyun geliştiriciliği tanıtılması hedeflenmiştir..



## **5.KAYNAKLAR**

1. GÜVENLİ İNTERNET MERKEZİ , Dijital Oyunlar Raporu 2019 , Ankara
  2. <https://en.wikipedia.org/wiki/Roguelike>
  3. Vojtech Cerny, Filip Dechterenko , 2018 , Rogue-Like Games as a Playground for Artificial Intelligence – Evolutionary Approach
  4. Darius Kazemi, Spelunky Generator Lessons , <http://tinysubversions.com/spelunkyGen/>
  5. IAN MILLIGTON , JOHN FUNGE , 2009 , ARTIFICIAL INTELLIGENCE FOR GAMES
- OYUNDA KULLANDIĞIM ASSETLERİN UNITY STORE LINKLERİ VE TASARIMCILARI**
6. GothicVania Church Pack , tasarımcı : Ansimuz ,  
<https://assetstore.unity.com/packages/2d/characters/gothicvania-church-pack-147117>
  7. Grotto Escape II , tasarımcı : Ansimuz ,  
<https://assetstore.unity.com/packages/2d/textures-materials/grotto-escape-ii-86689>
  8. 2D Pixel Item Asset Pack , tasarımcı : Startled Pixels ,  
<https://assetstore.unity.com/packages/2d/gui/icons/2d-pixel-item-asset-pack-99645>

## **YARARLANDIĞIM YOUTUBE TUTORIALLARI**

### **Blackthronprod kanalından,**

9. ROGUE LIKE RANDOM LEVEL GENERATION - INTERMEDIATE C#/UNITY TUTORIAL , <https://www.youtube.com/watch?v=hk6cUanSfXQ&t=8s>
10. HOW TO MAKE HEART/HEALTH SYSTEM - UNITY TUTORIAL - <https://www.youtube.com/watch?v=3uyolYVsiWc>
11. 2D PLATFORMER PATROL AI WITH UNITY AND C# , <https://www.youtube.com/watch?v=aRxuKoJH9Y0>

**Brackeys kanalından,**

12. MELEE COMBAT in Unity , <https://www.youtube.com/watch?v=sPiVz1k-fEs&t=1101s>

13. How to make a BOSS in Unity! ,

<https://www.youtube.com/watch?v=AD4JIXQDw0s&t=857s>

14. 2D Shooting in Unity , <https://www.youtube.com/watch?v=wkKsl1Mfp5M>

15. 2D PATHFINDING –

Enemy AI in Unity , <https://www.youtube.com/watch?v=jvtFUfJ6CP8>

## **6.ÖZGEÇMİŞ**

SEMA ŞEN

1997 İstanbul doğumlu. Cumhuriyet İlköğretim Okulunu bitirip liseyi Çapa Anadolu Öğretmen Lisesinde okudu. 2016 yılından beri İstanbul Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği bölümüne devam etmektedir.