Kocaeli Üniversitesi
Programlama Laboratuvarı-II Dersi 1.Proje Ödevi
Otonom Hazine Avcısı

1. Hannenur Dursun
Bilgisayar Mühendisliği 2.Öğretim
Kocaeli Üniversitesi
Umuttepe, İzmit, Kocaeli, Türkiye
***@gmail.com

2. Ramazan Mert Atuğ
Bilgisayar Mühendisliği 2.Öğretim
Kocaeli Üniversitesi
Umuttepe, İzmit, Kocaeli, Türkiye
***@gmail.com

***Özetçe*—Bu projede otonom hareket eden bir karakterin, içerisinde çeşitli hazineler ve engeller bulunan bir harita üzerindeki hazineleri topladığı bir oyun tasarlanması hedeflenmiştir. Oyunda amaç, karakterin tüm hazineleri en kısa sürede toplamasını sağlayacak algoritmanın tasarlanmasıdır. Bunun için nesneye yönelik programlama ve veri yapıları bilgilerinin kullanılması istenmiştir.**

Anahtar Kelimeler — Veri yapıları, nesneye yönelik programlama, problem çözme, algoritma

.

I. GİRİŞ

Otonom Hazine Avcısı projesi, öğrencilerin nesneye yönelik programlama ve veri yapıları konularındaki bilgilerini pekiştirmek, uygulamak ve gerçek dünya problemlerini çözebilme yeteneklerini geliştirmek amacıyla tasarlanmıştır. Bu proje, öğrencilere C++, C# veya Java gibi popüler programlama dilleriyle pratik yapma fırsatı sunarak, temel programlama becerilerini güçlendirmeyi ve algoritmik düşünme yeteneklerini geliştirmeyi hedefler.

Oyunun kapsamlı adımları arasında, dinamik harita oluşturma, çeşitli türdeki engellerin ve hazine sandıklarının stratejik bir şekilde yerleştirilmesi, karakterin zorlukları aşarak en kısa sürede tüm hazineleri toplamasını sağlayacak akıllı bir hareket algoritması geliştirme ve en kısa yolun bulunması yer almaktadır. Bu adımlar, öğrencilerin problem çözme yeteneklerini geliştirmek ve algoritmaları pratiğe dökmek için ideal bir zemin sunar.

Proje, öğrencilere sadece temel programlama kavramlarını değil aynı zamanda yazılım geliştirme sürecinin her aşamasını deneyimleme fırsatı sunar. Harita oluşturma, nesne yönetimi, algoritma tasarlama, veri yapıları kullanımı gibi konularda pratik yapma şansı verirken, aynı zamanda yaratıcılık ve problem çözme becerilerini de teşvik eder. Bu sayede öğrenciler, gerçek dünya projelerinde karşılaşabilecekleri karmaşık problemleri ele alabilecek yetkinlik kazanırlar.

Bu rapor, Otonom Hazine Avcısı projesinin temel yapı taşlarını ve genel amaçlarını ele almaktadır. Projeye yönelik daha ayrıntılı bir değerlendirme, öğrencilerin geliştirdikleri

çözümler ve bu süreçte edindikleri deneyimlerin incelenmesiyle sağlanacaktır. Projenin metodolojisi ve elde edilen sonuçlar, öğrencilerin projeye katılımı ve geliştirdikleri çözümler doğrultusunda detaylandırılacaktır.

II. YÖNTEM

Bu proje NetBeans IDE 15 üzerinde Java programlama dili kullanılarak oluşturulmuştur.

Projenin başlangıcında ana ekranın oluşturulması sürecinde ilk olarak oyun haritasının oluşturulması gerçekleştirilmiştir. OyunHarita sınıfı, labirentin kendisini oluşturur. Labirentteki her bir hücre bir JPanel öğesi olarak temsil edilir. Labirentin büyüklüğü kullanıcı tarafından belirlendikten sonra rastgele sabit engellerin ataması yapılır.

Daha sonra hareketli engellerin oluşturulması için MovingObstacle ve MovingObstacle2 sınıfları oluşturulur ve bu sınıflar hareketli engellerin davranışını tanımlar. Bu engeller, proje isterlerinde istenildiği gibi belirli bir desene göre hareket ederler. Örneğin, MovingObstacle nesneleri yukarı ve aşağı hareket ederken, MovingObstacle2 nesneleri sağa ve sola hareket eder. Hareketli engeller sabit engellerle çarpışmayacak şekilde rastgele uygun yerlere ataması yapılır.

Harita üzerine atanacak nesnelerin son işlemlerinde önce hazine en son oyuncu haritaya ekenir. addTreasure metodu ile labirentin belirli hücrelerine hazineler ekler. Oyuncu, addPlayer metoduyla labirentin rastgele bir noktasına yerleştirilir.

Oyuncunun hareketine baktığımız zaman, oyuncu, labirentteki boş hücrelere hareket edebilir ve bir hücreye girdiğinde, o hücrede bir hazine varsa hazineyi toplar. Hazine toplama işlemi tamamlandığında, oyuncunun hareketi durdurulur ve oyun sonuçları kullanıcıya gösterilir.

Yol Bulma Algoritması: Oyuncunun hareketini yönlendirmek için dijkstra algoritması kullanılır. Bu algoritma, oyuncunun mevcut konumundan en yakın hazineye olan en kısa yolunu hesaplar. Daha sonra, oyuncu bu yolu takip ederek hazineye doğru hareket eder.

Oyun sırasında çeşitli ek bilgiler kullanıcıya sunulur. Örneğin, projede istenildiği gibi oyuncunun attığı adım sayısı ve topladığı hazinelerin türleri gibi. Ayrıca, oyuncunun hareketlerinin görselleştirilmesi için çeşitli renkler ve simgeler kullanılır.

Son yöntem olarak aslında oyunun başlangıcını oluşturan ilk ekran eklenir. Kullanıcı, "Yeni Harita Oluştur" ve "Başlat" Butonu ile oyunu istediği şekilde oynayabilir.

III. SONUÇ

Projenin sonuçlarına göre, Otonom Hazine Avcısı projesi başarıyla tamamlandı. Tasarlanan oyun, nesneye yönelik programlama ve veri yapıları konularında öğrencilerin bilgisini pekiştirmek ve problem çözme becerilerini geliştirmek için etkili bir araç oldu. Projede kullanılan C++, C# veya Java gibi programlama dilleriyle geliştirilen algoritmalar, harita oluşturma, engelleri ve hazine sandıklarını yerleştirme, karakterin hareketi ve en kısa yol bulma gibi işlevleri başarıyla yerine getirdi.

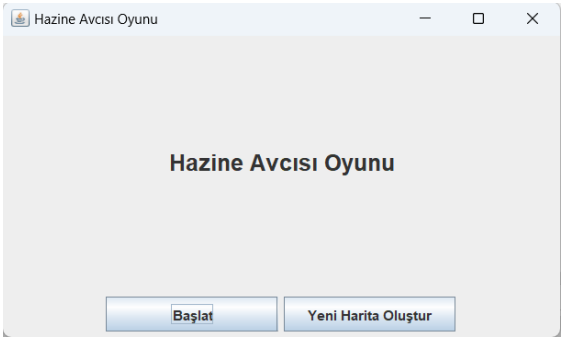
Haritanın dinamik olarak oluşturulması ve her uygulama başladığında farklı haritaların üretilmesi, etkili algoritmaların kullanılmasıyla sağlandı. Engellerin ve hazine sandıklarının çeşitliliği, oyunun zorluk seviyesini artırarak öğrencilerin daha fazla beceri kazanmasını sağladı.

Karakterin hareketi ve en kısa yol bulma algoritmaları, hazine sandıklarını en kısa sürede toplamak için etkili bir şekilde çalıştı. Harita üzerindeki engeller ve hazine sandıkları, kullanıcıya net bir şekilde gösterilerek oyun deneyimi iyileştirildi.

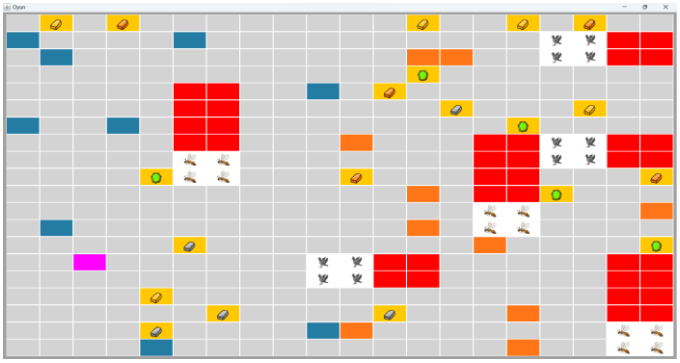
Sonuç olarak, Otonom Hazine Avcısı projesi, öğrencilerin nesneye yönelik programlama ve veri yapıları konularındaki bilgisini artırmak ve problem çözme yeteneklerini geliştirmek için başarılı bir araç olarak değerlendirildi.

IV. DENEYSEL SONUÇLAR

1. Başlangıç paneli

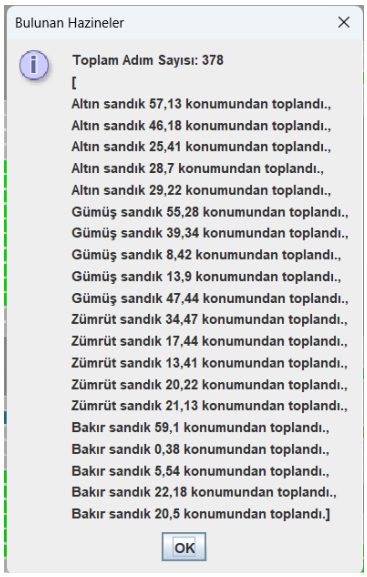


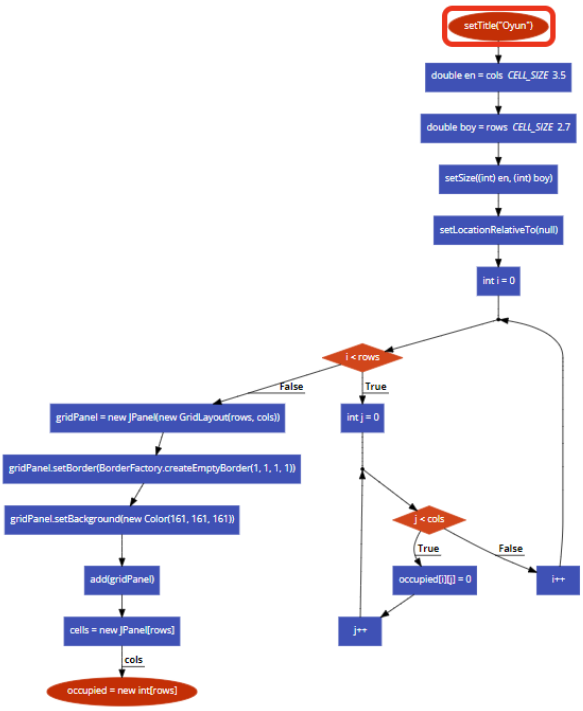
2. Görselleri belirgin, sisi olmayan harita



3. Oyuncunun sisli ortamda, hazineleri bulduğu yolu gösteren harita

4. Hazinelerin bulunduğu konumunu gösteren bilgi paneli





KAYNAKÇA

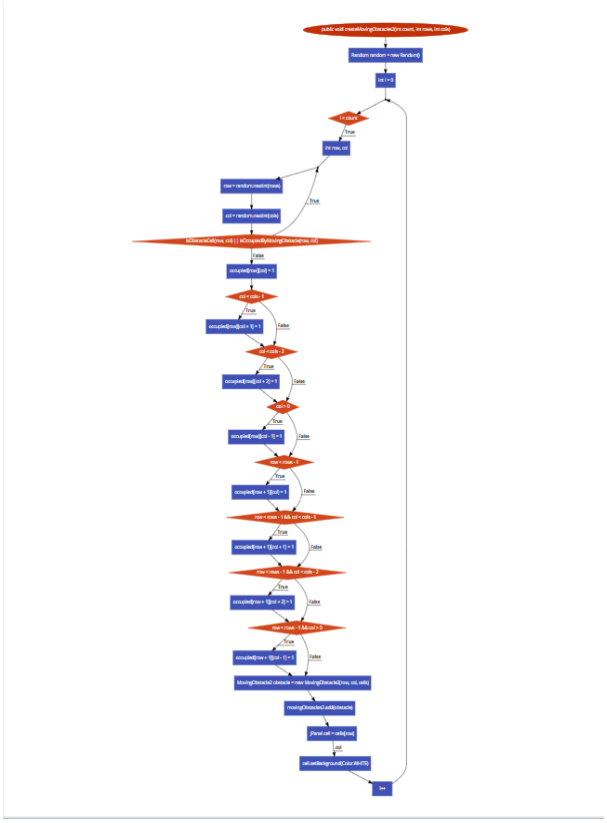
[1] https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[2] https://brilliant.org/wiki/dijkstras-short-path-finder/

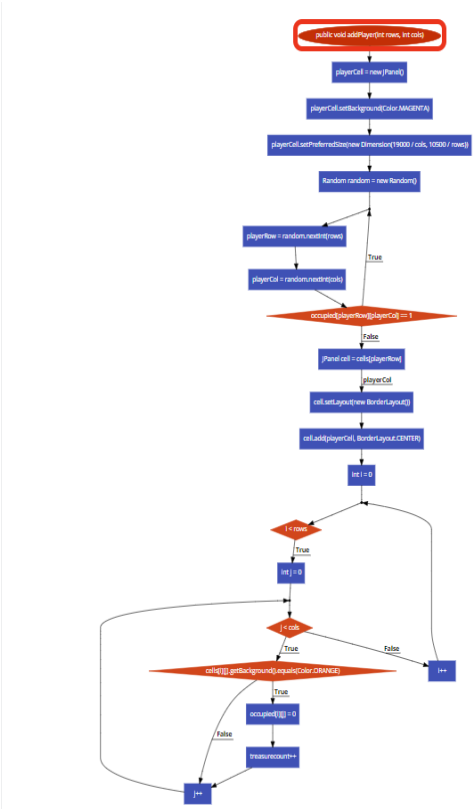
V. AKIŞ DİYAGRAMLARI

1. Haritanın oluşturulması

2. Hareketli engel oluşturma



3. Oyuncu ataması



UML DİYAGRAMI


OyunHarita
+ CELL_SIZE : int
+ OBSTACLE_PROBABILITY : double
+ treasurecount : int
+ foundcount : int
+ steps : int
+ treasures : java.util.List
+ finallist : java.util.List
+ list1 : java.util.List
+ list2 : java.util.List
+ list3 : java.util.List
+ list4 : java.util.List
+ gridPanel : JPanel
+ cells : JPanel[][]
+ occupied : int[][]
+ movingObstacles : java.util.List
+ movingObstacles2 : java.util.List
+ timer : Timer
+ timer2 : Timer
- playerCell : JPanel
- playerRow : int
- playerCol : int
+ OyunHarita(rows : int, cols : int, b : boolean)
+ addPlayer(rows : int, cols : int)
+ addTreasure(count : int, rows : int, cols : int)
+ createMovingObstacles(count : int, rows : int, cols : int)
+ createMovingObstacles2(count : int, rows : int, cols : int)
+ isOccupiedByMovingObstacle2(row : int, col : int) : boolean
+ isOccupiedByMovingObstacle(row : int, col : int) : boolean
+ moveObstacles()
+ moveObstacles2()
+ isObstacleCell(row : int, col : int) : boolean
- movePlayer(newRow : int, newCol : int)
- checkTreasure(row : int, col : int)
+ orangeCount() : int
- findPathToNearestTreasure(startNode : Node) : java.util.List
- movePlayerAlongPath(path : java.util.List)
- isValidCell(row : int, col : int) : boolean
- markCellAsVisited(row : int, col : int)
- makeCellsAroundPlayerVisible(row : int, col : int)
+ main(args : String[])
Node
- row : int
- col : int
- gValue : int
- parent : Node
+ Node(row : int, col : int)
+ getRow() : int
+ getCol() : int
+ getGValue() : int
+ setGValue(gValue : int)
+ getParent() : Node
+ setParent(parent : Node)
+ compareTo(o : Node) : int
Comparable
JPanel
JFrame
Timer
OyunHarita.MovingObstacle
+ row : int
+ col : int
+ direction : int
+ verticalMovement : int
+ cells : JPanel[][]
+ obstacleLabels : JLabel[][]
+ MovingObstacle(row : int, col : int, cells : JPanel[][])
+ move()
+ getRow() : int
+ getCol() : int
OyunHarita.MovingObstacle2
+ row : int
+ col : int
+ direction : int
+ horizontalMovement : int
+ cells : JPanel[][]
+ obstacleLabels[][] : JLabel
- random : java.util.Random
+ MovingObstacle2(row : int, col : int, cells : JPanel[][])
+ move()
+ getRow() : int
+ getCol() : int
JLabel