Nesne Yönelimli Programlama ile Otonom Savunma Sistemleri Simülasyonu: Kule Savunma Oyunu

Toprak ATEŞ
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
250201116@kocaeli.edu.tr

Melih Eren MALLI
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
250201112@kocaeli.edu.tr

***Özet*—Bu çalışma, nesne yönelimli programlama (NYP) prensipleri ve olay güdümlü mimari kullanılarak geliştirilen kapsamlı bir stratejik kule savunma (Tower Defense) simülasyonunu sunmaktadır. Proje, Java programlama dili ve JavaFX kütüphanesi kullanılarak, modüler, genişletilebilir ve interaktif bir oyun motoru tasarımını hedeflemektedir. Çalışma kapsamında, farklı özelliklere sahip düşman birimlerinin (polimorfizm) ve bu tehditleri bertaraf etmek için tasarlanan savunma kulelerinin (kalıtım) etkileşimleri modellenmiştir. Sistem, gerçek zamanlı grafiksel arayüz (GUI) ile arka plandaki simülasyon mantığının (backend) senkronize çalıştığı hibrit bir yapı üzerine kurulmuştur. Ayrıca, oyun içi olayların ve ekonomik dengelerin (para, can) kayıt altına alındığı bir 'savunma günlüğü' dosyalama sistemi entegre edilmiştir. Bu rapor, geliştirilen sistemin mimarisini, UML sınıf diyagramlarını, GUI-Simülasyon haberleşme protokollerini ve test sonuçlarını detaylandırmaktadır.**

***Index Terms*—JavaFX, Nesne Yönelimli Programlama, Oyun Motoru, Simülasyon, GUI, IEEE.**

I. GİRİŞ

A. Stratejik Simülasyonların Önemi

Bilgisayar oyunları ve simülasyonlar, karmaşık algoritmaların görselleştirilmesi ve kullanıcı etkileşimi ile birleştirilmesi açısından bilgisayar mühendisliği eğitiminde kritik bir rol oynamaktadır. Özellikle "Kule Savunma" (Tower Defense) türü, yol bulma (pathfinding), yapay zeka (AI) davranışları, kaynak yönetimi ve çarpışma tespiti (collision detection) gibi temel bilgisayar bilimleri problemlerini barındırması nedeniyle akademik çalışmalar için zengin bir zemin sunar.

B. Projenin Amacı ve Kapsamı

Bu projenin temel amacı, JavaFX platformu üzerinde çalışan, genişletilebilir bir oyun döngüsüne (Game Loop) sahip otonom bir savunma sistemi geliştirmektir. Proje, sadece görsel bir eğlence aracı olmanın ötesinde, NYP'nin temel taşları olan kapsülleme (encapsulation), kalıtım (inheritance) ve çok biçimlilik (polymorphism) kavramlarının somut bir uygulamasını sergilemeyi hedefler.

Geliştirilen simülasyon, "SpawnEnemy" sınıfı aracılığıyla dinamik olarak üretilen düşman dalgalarını, "SceneManager" sınıfı ile yönetilen sahne geçişlerini ve "FileManage" sınıfı ile gerçekleştirilen veri kalıcılığını (I/O) kapsamaktadır. Sistem, kullanıcıya stratejik kararlar alma imkanı tanırken, arka planda matematiksel hesaplamalarla (mesafe formülleri, hız vektörleri) oyun fiziğini işletmektedir.

II. OYUN KONSEPTİ VE MEKANİKLERİ

A. Konsept Seçimi ve Gerekçesi

Proje, "Füturistik/CyberPunk Kule Savunma" temasını benimsemiştir. Bu konseptin seçilme nedeni, farklı fiziksel özelliklerin (buzun yavaşlatması, lazerin hasar vermesi, zırhlı birliklerin dayanıklılığı) yazılım tarafında farklı sınıflar ve davranış ağaçları olarak modellenebilmesine olanak tanımasıdır. İsimlendirme stratejisi olarak, kod okunabilirliğini artırmak amacıyla İngilizce terimler (Tower, Enemy, Spawn) tercih edilmiş, ancak kullanıcı arayüzü ve çıktılar evrensel anlaşılırlık gözetilerek tasarlanmıştır.

B. Düşman Sınıflandırması

Simülasyonda üç temel düşman tipi modellenmiştir:

- **Leech (Standart):** Dengeli sağlık ve hıza sahip temel birimdir. Sistemin temel test verisidir.
- **Overlord (Zırhlı):** Yüksek sağlık puanına (HP) sahip ancak yavaş hareket eden birimdir. "Tank" sınıfı davranışı sergiler ve savunma hattını yarmak için kullanılır.
- **Bat (Uçan):** Zırhsız ancak çok hızlı hareket eden, bazı yer tabanlı kulelerin (Topçu Kulesi gibi) hedef alamadığı özel birimdir. Bu, oyuncuyu hava savunma stratejisi kurmaya zorlar.

C. Savunma Kuleleri ve Strateji

Oyuncu, sınırlı kaynaklarını (para) kullanarak aşağıdaki kuleleri stratejik noktalara yerleştirmelidir:

- **Laser Tower (Okçu/Lazer):** Düşük maliyetli, hızlı atış yapan, tekil hedefe odaklanan ve hem kara hem hava birimlerini vurabilen temel savunma birimidir.
- **Cannon Tower (Topçu):** Alan hasarı (Area of Effect - AoE) veren, ancak sadece kara birimlerine etki eden ağır savunma birimidir. Mermi tipi "cannonball" olarak tanımlanmıştır.
- **Snowball Thrower (Buz/Yavaşlatıcı):** Düşmanlara hasar vermekten ziyade, onların hareket hızını belirli bir süre (örneğin 3 saniye) %50 oranında düşüren stratejik destek birimidir.

III. YÖNTEM VE SİSTEM MİMARİSİ

Bu bölümde, yazılımın teknik altyapısı, kullanılan tasarım desenleri ve sınıf hiyerarşisi detaylandırılmıştır.

A. Yazılım Mimarisi ve Tasarım Desenleri

Proje, Model-View-Controller (MVC) mimarisine benzer bir yapıda kurgulanmıştır.

- **Singleton Deseni:** `SceneManager` sınıfı, uygulamanın tek bir örneği üzerinden tüm sahne geçişlerini (Menü, Ayarlar, Oyun, Bitiş) yöneterek bellek yönetimini optimize eder.
- **Factory Metodu:** `SpawnEnemy` sınıfı, gelen string parametrelerine (”standard”, ”flying”, ”armored”) göre dinamik olarak uygun düşman nesnesini bellekte oluşturur ve oyun sahnesine (Pane) ekler.
- **Game Loop (Oyun Döngüsü):** JavaFX `Timeline` ve `KeyFrame` yapıları kullanılarak, saniyede belirli kare sayısında (FPS) ekranın güncellenmesi sağlanmıştır.

```
org.example.towerproject
 base
    Bullet.java
    Enemy.java (Abstract)
    Tile.java
    Tower.java (Abstract)
 enemies
    Bat.java
    Leech.java
    Overlord.java
 filemanage
    FileManage.java
 game
    Game.java
    Gamer.java
 scenes
    endmenu
    playthrough
       enemymanager
       towermanager
    settings
    startmenu
    SceneManager.java
    Scenes.java
 towers
     CanonTower.java
     LaserTower.java
     SnowballThrower.java
```

Şekil 1. Proje Dosya ve Paket Hiyerarşisi

B. Sınıf Hiyerarşisi (UML Analizi)

Sistemdeki temel yapı taşları soyut sınıflar (abstract class) üzerine kurulmuştur:

1) Enemy Sınıfı: Tüm düşmanlar `Enemy` soyut sınıfından türetilir. Bu sınıf; `pathTransition` (hareket animasyonu), `currentHealth`, `speed` ve `armor` gibi ortak özellikleri barındırır. `takeDamage()` metodu, zırh hesaplamasını yaparak hasarı işler.

2) Tower Sınıfı: Tüm kuleler `Tower` soyut sınıfını genişletir. Kulelerin `attack()` metodu her kule tipi için özelleşmiştir. Kuleler, menzillerine giren düşmanları Öklid mesafesi formülü ($d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$) ile tespit eder.

C. GUI ile Simülasyon Motoru Haberleşmesi

Grafik Arayüz (GUI) ile mantık katmanı arasındaki haberleşme, ”Event-Driven” (Olay Güdümlü) bir yaklaşımla sağlanmıştır.

1) **Kullanıcı Girdisi:** Oyuncu, `GridPane` üzerindeki bir hücreye tıkladığında, `PlaceTower` sınıfı bu tıklamanın koordinatlarını yakalar.
2) **Doğrulama:** Tıklanan alanın bir ”Yol” (Path) olup olmadığı ve oyuncunun yeterli parası olup olmadığı kontrol edilir.
3) **Görselleştirme:** Eğer işlem geçerliyse, `gamePane` üzerine dinamik olarak bir `ImageView` eklenir ve kule aktif hale getirilir.
4) **Animasyon Tetikleme:** Kule yerleştirildiği andan itibaren kendi içindeki `Timeline`'ı başlatır ve menziline giren düşmanları taramaya başlar. Düşman tespit edildiğinde `Bullet` nesnesi oluşturulur ve bir animasyon ile hedefe yönlendirilir.

IV. SİMÜLASYON VE BULGULAR

A. Test Senaryosu ve Dalga (Wave) Yapısı

Simülasyon, giderek zorlaşan dalga senaryoları üzerinde test edilmiştir. `PlaythroughController` sınıfında tanımlanan bu dalglar şu şekildedir:

- **Dalga 1:** Ağırlıklı olarak standart (Leech) düşmanlardan oluşur, araya serpiştirilmiş zırhlı birimler bulunur.
- **Dalga 2:** Yoğun uçan (Bat) ve zırhlı (Overlord) düşman akını içerir. Bu dalga, oyuncunun hem hava savunmasını hem de yüksek hasar (DPS) kapasitesini test eder.

B. Savunma Günlüğü (Logging) Analizi

Oyun sırasında gerçekleşen kritik olaylar, `FileManage` sınıfı aracılığıyla `savunma_gunlugu.txt` (kod içinde gamelog.txt olarak geçer) dosyasına zaman damgası ile kaydedilir.

Tablo I
SAVUNMA GÜNLÜĞÜ ÖRNEK ÇIKTILARI

Zaman	Olay Açıklaması
14:30:01	TOWER GAME PROJECT - STARTED
14:30:15	A Cannon Tower placed on: 275.0, 125.0
14:30:45	One enemy is killed after taking damage: 20
14:31:00	Money accumulation has reached 300$
14:31:20	One bat has reached the finish line
14:32:10	Waves Complete - Game Won by Player

Bu loglar, oyunun dengesini (balancing) analiz etmek için kullanılmıştır. Örneğin, oyuncunun parası çok hızlı tükeniyorsa veya düşmanlar hiç ölmeden bitişe ulaşıyorsa, kule hasar değerleri bu çıktılara göre revize edilmiştir.

C. Performans Değerlendirmesi

Yapılan testlerde, ekranda aynı anda 20'den fazla hareketli nesne (düşman ve mermi) bulunduğunda dahi JavaFX `Timeline` yapısının 60 FPS akıcılığını koruduğu gözlemlenmiştir. Çarpışma testleri (Hitbox detection), her karede (frame) tüm düşmanlar ve mermiler arasında değil, sadece menzil içindeki nesneler arasında yapılarak işlemci yükü optimize edilmiştir.

Şekil 2. Simülasyon anında alınan ekran görüntüsü.

Örneğin, buz kulesi (SnowballThrower) bir düşmanı vurduğunda, düşmanın `pathTransition` hızı (rate) dinamik olarak 0.5'e düşürülmüş, 3 saniye sonra tekrar 1.0'a çekilmiştir. Bu durum, olay güdümlü programlamanın (zamanlayıcılar ile) başarıyla uygulandığını göstermektedir.

V. SONUÇ VE GELECEK ÇALIŞMALAR

A. Sonuç

Bu proje kapsamında, modüler ve nesne yönelimli bir kule savunma oyunu motoru başarıyla geliştirilmiştir. `Gamer` sınıfı üzerinden oyuncunun sağlık ve ekonomi yönetimi sağlanmış, `PlaceTower` sınıfı ile dinamik grid etkileşimi kurulmuştur. Elde edilen simülasyon, "Bat" gibi uçan birimlerin kara kuleleri tarafından vurulamaması gibi kompleks oyun mantıklarını (game logic) hatasız bir şekilde işleyebilmektedir. `savunma_gunlugu.txt` çıktıları, sistemin deterministik yapısını ve olay takibini doğrularken, görsel arayüz kullanıcıya anlık geri bildirim sağlamıştır.

B. Kısıtlar ve Gelecek Çalışmalar

Mevcut sistemde harita "TileCoord" enum yapısı üzerinde statik olarak tanımlanmıştır. Gelecek çalışmalarda, A* (A-Star) gibi yol bulma algoritmaları entegre edilerek, oyuncunun kuleleri labirent oluşturacak şekilde yerleştirmesine ve düşmanların dinamik olarak yol bulmasına olanak sağlanabilir. Ayrıca, XML veya JSON tabanlı bir seviye editörü eklenerek, kod tekrar derlenmeden yeni bölümlerin tasarlanması mümkün kılınabilir.

VI. YAZAR KATKILARI

Projenin geliştirilmesi sürecinde yazarların sorumluluk alanları aşağıdaki gibidir:

- **Toprak ATEŞ (250201112):** Projenin mimarı olarak tüm yazılım geliştirme süreçlerini üstlenmiştir. Oyun motorunun çekirdeği (Game Loop), Singleton ve Factory tasarım desenlerinin uygulanması, düşman yapay zekası (AI), kule mekanikleri, çarpışma algoritmaları ve dosya yönetim sisteminin (FileManage) kodlanması tarafınca gerçekleştirilmiştir.
- **Melih Eren MALLI (250201116):** Projenin görsel estetiği ve tema tasarımından sorumlu olmuştur.

KAYNAKLAR

[1] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Language Specification*, 3rd ed. Addison-Wesley, 2005.

[2] A. Tornhill, *Lisp for the Web*, Leanpub, 2014. (Oyun döngüsü mantığı referansı).

[3] Oracle Documentation, "JavaFX Architecture," [Online]. Available: https://docs.oracle.com/javafx/. [Erişim Tarihi: Aralık 2025].

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994. (Singleton ve Factory desenleri referansı).

[5] T. Akenine-Möller, E. Haines, N. Hoffman, *Real-Time Rendering*, 4th ed. CRC Press, 2018. (Oyun fiziği ve çarpışma tespiti referansı).