

Kuş Bakışı Atış Oyunu (ZombieShot)

BİL 211 - Nesne Yönelimli Programlama

Oğuz Nurlu (211101069)

1 Oyun Hakkında

Bu projede Java Swing kullanarak kuş bakışı bir atış oyunu (top-down shooter) geliştirilmiştir. Oyuncu, dalga dalga gelen zombilerden hayatı kalmaya çalışırken fare ile hedef alarak çeşitli silahlar kullanır. Her dalga ile birlikte zombilerin sayısı ve çeşitliliği artarak oyunun zorluk seviyesi yükselir.

2 Kurulum ve Oynanış Kılavuzu

2.1 Kurulum

Oyunu derlemek ve çalıştmak için aşağıdaki adımları izleyin:

1. Kaynak kodunu içeren `source.zip` dosyasını çıkarın
2. Komut satırında projenin ana dizinine gidin
3. Aşağıdaki komutları sırasıyla çalıştırın:

```
1 javac *.java  
2 java Game
```

2.2 Oynanış Kontrolleri

- **W, A, S, D:** Karakteri hareket ettirme
- **Fare İmleci:** Nişan alma
- **Sol Fare Tuşu:** Ateş etme

- **R:** Silahı yeniden doldurma (reload)
- **1-5:** Silah değiştirme (silah mermisi kazandıkça kullanılabilirler)
- **ESC:** Oyunu duraklatma

2.3 Temel Mekanikler

- Oyuncu, fare imleci ile nişan alarak zombileri öldürmeye çalışır
- Belirli dalgalar tamamlandığında yeni silahlar zombilerden düşebilir açılır
- Zombilerden sağlık ve mermi takviyesi düşebilir
- Öldürülen her zombi için puan kazanılır
- Oyuncu öldüğünde oyun sona erer

3 Sınıf Yapısı ve OOP Gerçeklemeleri

3.1 Ana Oyun Sınıfları

3.1.1 GameFrame

Ana oyun penceresi olarak görev yapar ve temel bileşenleri içerir. JFrame'den türeyen bu sınıf, oyunun görsel arayüzünün temelini oluşturur.

3.1.2 GamePanel

JPanel'den türeyen bu sınıf, oyunun ana mantığını yönetir ve ActionListener arayüzüünü gerçekler. Oyun döngüsü, çarpışma tespiti, zombi ve mermi yönetimi bu sınıfta bulunur.

```

1 public class GamePanel extends JPanel implements
2     ActionListener {
3     // Oyun mantığını isleyen ana döngü
4     @Override
5     public void actionPerformed(ActionEvent e) {
6         // Oyun durumunu güncelle
7         // Çarpışmaları kontrol et
8         // Oyun nesnelerini güncelle

```

```
8     }
9 }
```

Listing 1: GamePanel’ın temel yapısı

3.1.3 GameInfo

Oyunun durumunu, constant değişkenleri tutan ve veri depolayan bir sınıfıtır. Oyun kaydetme ve yükleme işlemlerinden sorumludur ve tüm oyun verilerini merkezileştirir.

3.1.4 StatPanel

Oyun istatistiklerini (can, mermi sayısı, skor, dalga bilgisi) gösteren panel sınıfı.

3.2 Kalıtsallık ve Polimorfizm Örnekleri

3.2.1 Entity Sınıf Hiyerarşisi

Entity soyut sınıfı, oyundaki tüm dinamik nesnelerin temelini oluşturur. Konum, boyut ve çarpışma alanı gibi tüm nesnelerin ortak özelliklerini içerir.

3.2.2 Zombi Türleri

Zombie soyut sınıfını temel alan dört farklı zombi türü bulunmaktadır. Her biri farklı davranışlar ve özellikler sergilemektedir:

- **NormalZombie:** Temel zombi tipi, standart davranış
- **ReptileZombie:** Hızlı ve düşük canlı, oyuncunun yakınına geldiğinde ona doğru zıplama özelliği
- **TankZombie:** Yüksek canlı, yavaş hareket eder, normal hasar verir
- **AcidicZombie:** Uzaktan asit atabilir, öldüğünde çevreye hasar verir

Polimorfizm sayesinde tüm zombi tipleri, aynı arayüz üzerinden farklı davranışlar sergileyebilmektedir:

```

1 // GamePanel içinde zombilerin güncellenmesi
2 for (Zombie zombie : gameInfo.zombies) {
3     if (zombie instanceof ReptileZombie) {
4         ReptileZombie reptileZombie = (ReptileZombie)zombie;
5         if (reptileZombie.isJumping) {
6             // Ziplama davranışları
7         }
8     } else if (zombie instanceof AcidicZombie) {
9         // Asit atma davranışları
10    }
11    // Genel zombi davranışları
12 }
```

Listing 2: Zombi polimorfizm örneği

3.2.3 Silah Sistemi

Weapon soyut sınıfından türeyen beş farklı silah türü bulunmaktadır:

- **Pistol:** Düz atış, sapma yok, sınırsız yedek şarjör
- **Rifle:** Yüksek atış hızı, 30 derece sapma
- **Shotgun:** 9 mermi içeren 45 derecelik saçılma
- **Sniper:** Zombileri delip geçen mermiler
- **RocketLauncher:** Patlama etkisi ile toplu hasar

Kalıtsallık ve polimorfizm sayesinde tüm silahlar aynı temel arayüzü kullanmakta, ancak farklı davranışlar sergilemektedir.

3.3 Arayüz (Interface) Kullanımı

ActionListener arayüzü, oyun döngüsünü kontrol etmek için kullanılmış ve Timer nesneleriyle birleştirilmiştir. GamePanel, bu arayüzü gerçekleyerek düzenli olarak oyun durumunu günceller.

3.4 Nesnelerin Yeniden Kullanımı - Object Pooling

Mermi (Bullet) nesneleri, performansı artırmak için bir nesne havuzu (object pool) tasarımlı ile yönetilmektedir. Bu, sürekli yeni nesnelerin oluşturulması yerine, önceden oluşturulan nesnelerin yeniden kullanılmasını sağlar.

```
1 // Bullet sınıfı içinde static bir nesne havuzu
2 public static class Pool {
3     private static final List<Bullet> pool = new ArrayList
4         <>();
5
6     public static Bullet getBullet(double x, double y, Weapon
7         weapon) {
8         // Havuzda varsa kullan, yoksa yeni oluştur
9     }
10
11    public static void returnBullet(Bullet bullet) {
12        // Kullanım sonrası havuza geri dondur
13    }
14 }
```

Listing 3: Mermi havuzu örneği

4 Öne Çıkan Özellikler ve Ek Geliştirmeler

4.1 Görsel Efektler ve Optimizasyonlar

4.1.1 Görüntü İşleme Efektleri

Oyun, karakter hasar aldığında ya da zombilere hasar verdiğiinde flash efekti kullanmaktadır. Bu efekt, ColorConvertOp sınıfı ile görüntünün gri tonlara dönüştürülmesiyle gerçekleştirilmiştir:

```
1 private ColorConvertOp flashEffect = new ColorConvertOp(
2     ColorSpace.getInstance(ColorSpace.CS_GRAY), null);
3
4 // Kullanım ornegi
5 if (zombie.isFlashing()) {
6     BufferedImage flashImage = flashEffect.filter(zombie.
7         image, null);
8     g2d.drawImage(flashImage, 0, 0, zombie.width, zombie.
9         height, null);
10 }
```

Listing 4: Flash efekti gerçeklemesi

4.1.2 Görüntü Önbellekleme (Image Caching)

Sürekli tekrarlanan görüntü dönüştürmelerini önlemek için önbellekleme tekniği kullanılmıştır:

```
1 // Once'den hesaplanmış flash goruntuleri
2 private BufferedImage playerFlashImage = null;
3 private BufferedImage[] zombieFlashImages = new BufferedImage
4     [4];
5
6 // Önbellekleme ile kullanım
7 if (useImageCaching && zombieFlashImages[zombieType] == null)
8 {
9     zombieFlashImages[zombieType] = flashEffect.filter(zombie
10 .image, null);
11 }
```

Listing 5: Görüntü önbellekleme

4.1.3 Görüntü Alanı Kırpma (Viewport Culling)

Ekranda görülmeyen nesnelerin gereksiz yere çizilmesini önlemek için Viewport Culling teknigi uygulanmıştır:

```
1 private boolean isEntityVisible(Entity entity, int minX, int
2 minY,
3                                     int maxX, int maxY) {
4     if (!useViewportCulling) return true;
5
6     return entity.x + entity.width >= minX && entity.x <=
7         maxX &&
8             entity.y + entity.height >= minY && entity.y <=
9                 maxY;
10 }
11
12 // Cizim sirasinda kullanim
13 if (isEntityVisible(zombie, viewportMinX, viewportMinY,
14                     viewportMaxX, viewportMaxY)) {
15     // Zombiyi ciz
16 }
```

Listing 6: Görüntü alanı kırpma

4.2 Oyun İçi Ekstra Özellikler

4.2.1 Zombi Yön Göstergeleri

Ekran dışındaki zombilerin yerini göstermek için yön göstergeleri eklenmiştir:

```
1 private void drawZombieIndicators(Graphics2D g2d) {  
2     // Ekran disindaki zombiler icin yon oklari ciz  
3 }
```

Listing 7: Zombi yön göstergeleri

4.2.2 Silah Soğuma ve Şarjör Doldurma Çubukları

Oyuncuya görsel geri bildirim sağlamak için silah soğuma ve şarjör doldurma ilerleme çubukları eklenmiştir:

```
1 private void drawCooldownBar(Graphics2D g2d) {  
2     // Silah soguma cubugunu ciz  
3 }  
4  
5 private void drawReloadingBar(Graphics2D g2d) {  
6     // Sarjor doldurma cubugunu ciz  
7 }
```

Listing 8: Silah ilerleme çubukları

4.3 Veri Yönetimi

Oyun durumunu kaydetme ve yükleme özelliği, Serializable arayüzü kullanılarak gerçekleştirilmiştir. Bu sayede oyun durumu istendiğinde kaydedilebilir ve daha sonra yüklenebilir. Oyuncu isterse, Game Over menüsünde en son kaydedilen save’ı tekrar yükleyebilir. Save’ler otomatik alınmıyor fakat oyuncu kendisi aldıysa o save’e geri dönebilir.

4.4 Ses Sistemi

Oyunda silah sesleri, zombi sesleri ve arka plan müziği için bir ses sistemi geliştirilmiştir. Birden fazla arkaplan müziği vardır ve oyun içerisinde bu müzikler rastgele sırayla çalar.

```
1 public class MusicPlayer {
2     private static Clip clip;
3
4     public static void playMusic(String filePath) {
5         // Muzik calma
6     }
7
8     public static void dispose() {
9         // Muzik kaynaklarini temizleme
10    }
11 }
```

Listing 9: Ses sistemi

4.5 Harita

Oyunda harita, sürekli rastgele olusacak şekilde tasarlanmıştır. Oyuncular ve zombiler ağaçlara takılabilirler fakat mermileri bu ağaçların içerisinde geçebilir. Bunun için Background class'ı yazılmıştır.

```
1 public class Background implements Serializable {
2     protected final int TILE_SIZE = 64;
3     private final double OBSTACLE_PROBABILITY = 0.25;
4
5     public Background(String[] backgroundTileImages, String[]
6         obstacleTilePaths) {
7         // Yeni bir rastgele harita olustur
8     }
9
10    public boolean isValidMove(double dx, double dy, int
11        playerHeight) {
12        // Karakterin buraya hareket edip edemeyecegini don
13    }
14 }
```

Listing 10: Ses sistemi

5 Oyun Ekran Görüntüleri

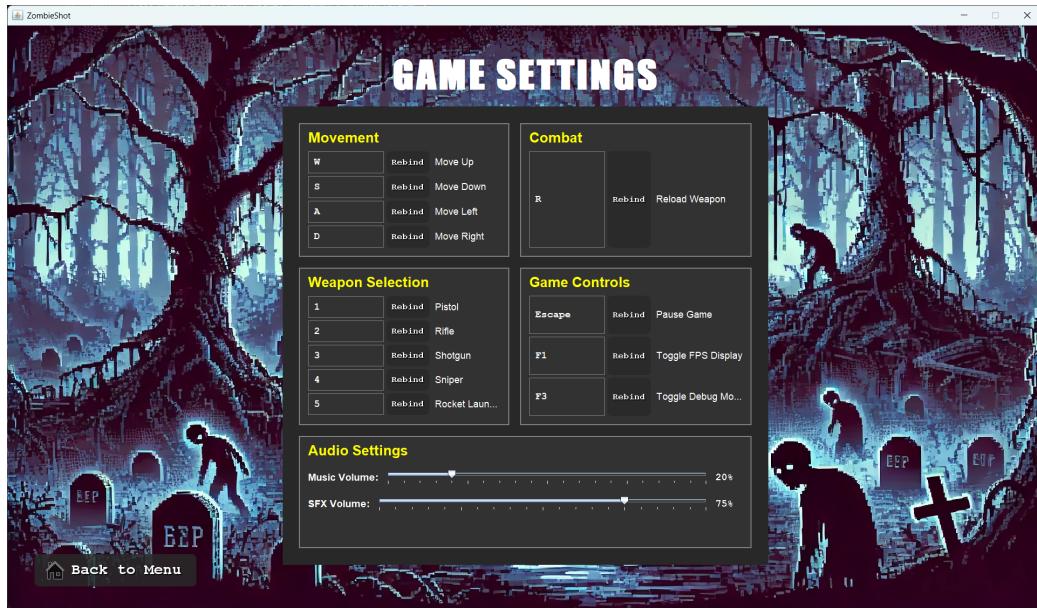
5.1 Ana Menü



Şekil 1: Oyun Ana Menüsü

New Game ve New Game+ modları vardır. New Game+ modunda zombiler daha fazla ve daha hızlı canlanır, silahların soğuma süreleri daha azdır ve daha fazla mermileri vardır. Oyunun performansını ölçmek için kullanılabilir. Load game ile kaydedilen oyun başlatılabilir, ayarlar menüsüne girilebilir veya oyundan çıkışılabilir.

5.2 Ayarlar Ekranı



Şekil 2: Ayarlar Ekranı

Her keybind değiştirilebilir şekilde tasarlanmıştır ve bu menüden istediğimiz ayarı değiştirebiliriz. FPS sayacı aktif edildiğinde sağ üstte küçük beyaz bir şekilde gözükmür. Debug menüsü ise zombilerin, karakterlerin veya arkaplanların collision'ları gibi özellikleri görmemizi sağlar.

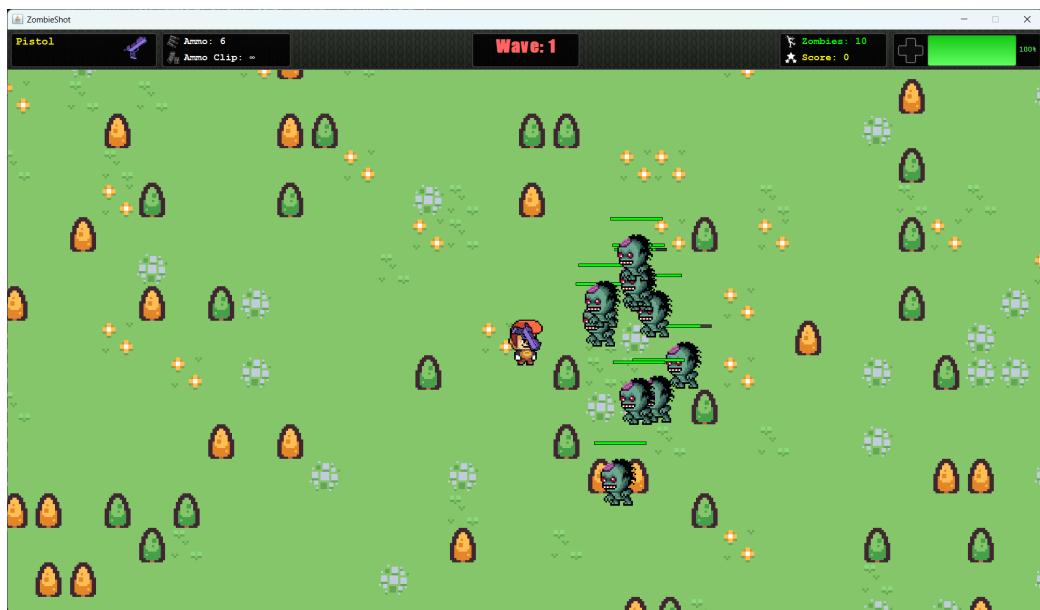
5.3 Karakter Seçim Ekranı



Şekil 3: Karakter Seçim Ekranı

12 karakter içerisinde istedigimizi seçip oyuna başlayabiliriz.

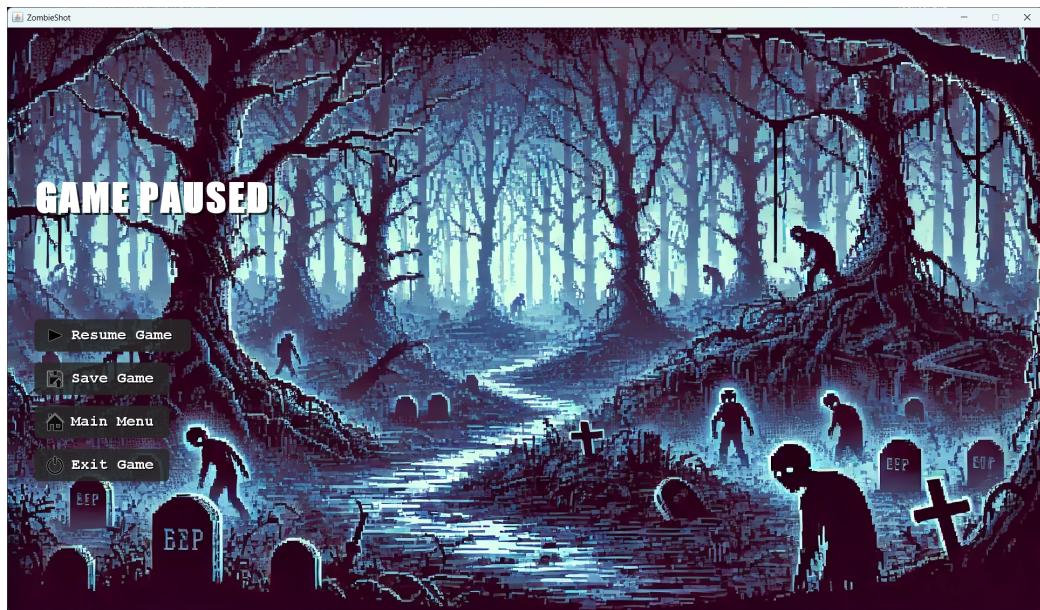
5.4 Oyun Ekranı



Şekil 4: Oynanış Görüntüsü

Sol üstte silahın ismi, ikonu, şarjördeki mermi ve toplam mermi miktarı gözükmektedir. Orta üstte kaçinci dalgada olduğumuz vardır. Sağ üstte de şu an canlı olan zombi sayısı, toplam skorumuz ve canımız gözükmektedir.

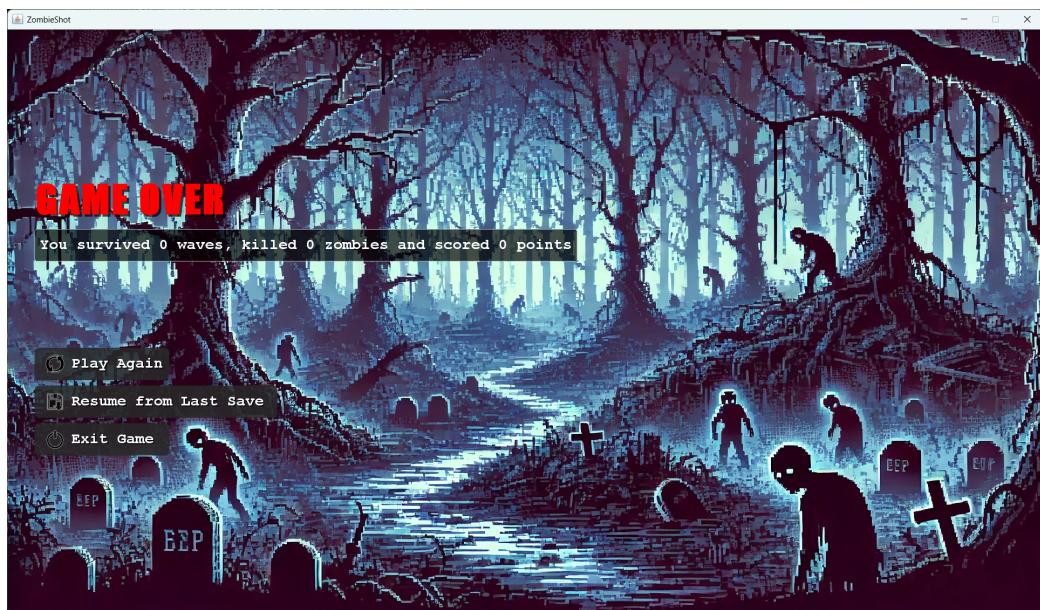
5.5 Duraklatma Ekranı



Şekil 5: Duraklatma Ekranı

Kullanıcı burada oyunu devam ettirebilir, kaydedebilir, ana menüye dönebilir veya oyundan çıkabilir.

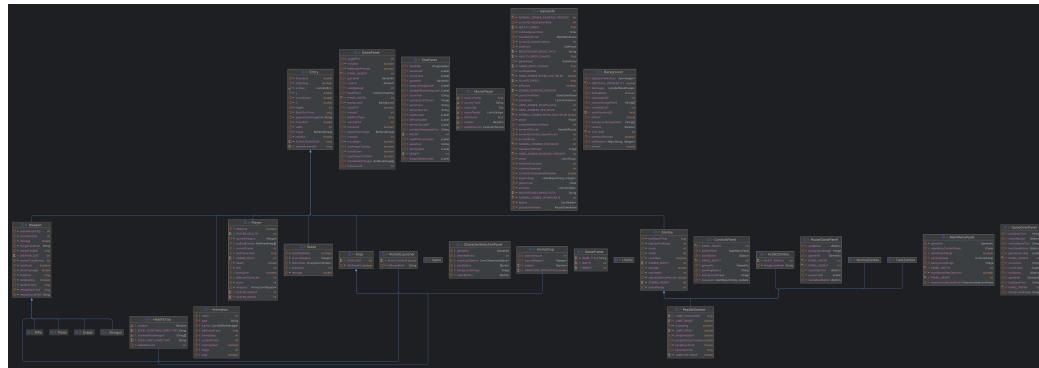
5.6 Oyun Sonu Ekranı



Şekil 6: Oyun Sonu Ekranı

Burada oyun sonu istatistiklerini görüp oyunu tekrar başlatabilir, oyun kaydedildiyse son kayıta geri dönülebilir veya oyundan çıkışılabilir.

6 UML Diyagramı



Şekil 7: UML Diyagramı

7 Sonuç

Bu projede, Java Swing kullanarak kapsamlı bir kuş bakışı atış oyunu geliştirilmiştir. Projede nesne yönelimli programlama ilkeleri (kalıtsallık, polimorfizm, kapsülleme ve soyutlama) uygulanmış, performans ve oynanabilirlik için çeşitli optimizasyon teknikleri kullanılmıştır.

Oyun, zombilerin davranışları, silahların çeşitliliği ve oyun içi özellikle-riyle zengin bir deneyim sunmaktadır. Veri yönetimi ve oyun durumu kay-detme özelliklerinin yanı sıra görsel efektler ve ses sistemi ile oyun deneyimi geliştirilmiştir.