Breakout Oyunu - Teknik Rapor

Genel Bakış

Bu proje, FreeGLUT ve OpenGL kullanılarak geliştirilmiş klasik bir Breakout (Tuğla Kırma) oyunudur. Oyun, C++ programlama dili ile yazılmış olup, temel 2D grafik işlemleri ve basit fizik simülasyonları içermektedir.

Oyun Mimarisi

1. Temel Bileşenler

1.1 Veri Yapıları Vector2 Struct:

```
struct Vector2 {
    float x, y;
    Vector2(float x = 0, float y = 0) : x(x), y(y) {}
    Vector2 operator+(const Vector2& other) const;
    Vector2 operator*(float scalar) const;
};
```

- 2D pozisyon ve hız vektörlerini temsil eder
- Operatör aşırı yükleme ile matematiksel işlemler kolaylaştırılmıştır
- Vektör toplama ve skaler çarpım işlemleri desteklenir

Oyun Nesneleri: - **Brick (Tuğla):** Pozisyon, aktiflik durumu ve renk bilgisini içerir - **Ball (Top):** Pozisyon ve hız vektörlerini içerir - **Paddle (Raket):** Sadece pozisyon bilgisini içerir

2. Oyun Döngüsü ve Zaman Yönetimi

2.1 Delta Time Hesaplaması

```
int currentTime = glutGet(GLUT_ELAPSED_TIME);
float deltaTime = (currentTime - lastTime) / 1000.0f;
lastTime = currentTime;
```

- Kare bağımsız hareket için delta time kullanılır
- Milisaniye cinsinden zaman farkı saniyeye çevrilir
- Bu sayede farklı FPS değerlerinde tutarlı hareket sağlanır

2.2 Ana Oyun Döngüsü

1. **Input İşleme:** Klavye girişleri kontrol edilir
2. **Fizik Güncelleme:** `updateGame(deltaTime)` fonksiyonu çağrılır
3. **Render:** Tüm oyun elemanları çizilir
4. **Buffer Swap:** Double buffering ile görüntü güncellenir

3. Fizik ve Hareket Algoritmaları

3.1 Hareket Hesaplaması

```
ball.position = ball.position + ball.velocity * deltaTime;
```

- Euler entegrasyonu kullanılır
- Pozisyon = Eski Pozisyon + (Hız × Zaman Farkı)
- Frame rate'den bağımsız düzgün hareket sağlar

3.2 Çarpışma Algılama - AABB (Axis-Aligned Bounding Box)

```
bool checkCollision(const Vector2& pos1, float w1, float h1,
                    const Vector2& pos2, float w2, float h2) {
    return pos1.x < pos2.x + w2 &&
           pos1.x + w1 > pos2.x &&
           pos1.y < pos2.y + h2 &&
           pos1.y + h1 > pos2.y;
}
```

- Dikdörtgen-dikdörtgen çarpışma testi
- X ve Y eksenleri ayrı ayrı kontrol edilir
- Tüm koşullar sağlandığında çarpışma var demektir

3.3 Top-Raket Çarpışma Fiziği

```
float paddleCenter = paddle.position.x + PADDLE_WIDTH / 2;
float ballCenter = ball.position.x + BALL_SIZE / 2;
float hitPos = (ballCenter - paddleCenter) / (PADDLE_WIDTH / 2); // -1 to 1
ball.velocity.x = hitPos * BALL_SPEED;
```

- Topun raketin neresine çarptığı hesaplanır (-1 ile 1 arası)
- Çarpma noktasına göre yansıma açısı belirlenir
- Merkeze yakın çarpışmalar daha dik açıyla yansır

3.4 Hız Normalizasyonu

```
float speed = sqrt(ball.velocity.x * ball.velocity.x +
                   ball.velocity.y * ball.velocity.y);
ball.velocity.x = (ball.velocity.x / speed) * BALL_SPEED;
ball.velocity.y = (ball.velocity.y / speed) * BALL_SPEED;
```

- Vektör büyüklüğü (hız) Pisagor teoremi ile hesaplanır
- Hız vektörü normalize edilerek sabit hız korunur
- Bu sayede top her zaman aynı hızda hareket eder

4. Render Sistemi

4.1 OpenGL 2D Projeksiyon Ayarı

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT, -1, 1);
```

- Ortografik projeksiyon kullanılır (2D oyun için ideal)
- Koordinat sistemi: Sol alt (0,0), sağ üst (800,600)
- Z ekseni -1 ile 1 arasında (2D için yeterli)

4.2 Geometrik Şekil Çizimi Dikdörtgen Çizimi: - `GL_QUADS` primitifi kullanılır - 4 köşe noktası saat yönünde tanımlanır

Daire Çizimi:

```
glBegin(GL_TRIANGLE_FAN);
for (int i = 0; i <= 20; i++) {
    float angle = 2.0f * 3.14159f * i / 20;
    glVertex2f(x + cos(angle) * radius, y + sin(angle) * radius);
}
```

- Triangle fan tekniği ile daire yaklaşımı
- 20 üçgen kullanılarak düzgün görünüm sağlanır
- Trigonometrik fonksiyonlarla nokta koordinatları hesaplanır

5. Oyun Durumu Yönetimi

5.1 Durum Değişkenleri

- `gameRunning`: Oyun aktif mi?
- `gameWon`: Oyun kazanıldı mı?
- `gameLost`: Oyun kaybedildi mi?
- `currentLevel`: Mevcut seviye numarası

5.2 Seviye Yönetimi

```
if (currentLevel == 1) {
    // Düz tuğla dizilimi
} else if (currentLevel == 2) {
    // Dama tahtası deseni
    if ((col + row) % 2 == 0) {
        // Tuğla ekle
    }
}
```

- Her seviye için farklı tuğla dizilimleri
- Modülo operatörü ile desen oluşturma
- Dinamik seviye yükleme altyapısı

6. Giriş İşleme Sistemi

6.1 Klavye Durumu Takibi

```
bool keys[256] = {false};
```

- Her tuş için boolean durum saklanır
- `processInput`: Tuşa basıldığında true
- `processInputUp`: Tuş bırakıldığında false
- Bu sistem sürekli hareket için idealdir

7. Hata Yönetimi ve Loglama

```
void checkOpenGLError(const char* operation) {
    GLenum error = glGetError();
    if (error != GL_NO_ERROR) {
        // Hata logla
    }
}
```

- OpenGL hatalarının sistematik kontrolü
- Detaylı hata mesajları log dosyasına yazılır
- Debug için önemli bilgiler saklanır

Sonuç

Bu Breakout oyunu, temel oyun programlama konseptlerinin başarılı bir uygulamasıdır. AABB çarpışma algılama, delta time tabanlı hareket, basit fizik simülasyonu ve verimli render teknikleri kullanılmıştır. Kod yapısı modüler ve genişletilebilir olup, yeni seviyeler ve özellikler kolayca eklenebilir.