Flappy Bird Open GL / C++

Ömer Ertekin

Muhammed Erdoğan

Oyun geliştirmeye olan merakımızdan dolayı proje konusu olarak eskiden oldukça popüler olan flappy bird oyununu yapmaya karar verdik. Bu oyunda yapmamız gereken aşamaları sırasıyla bu adımlar oldu

1- Playerı çizdirme ve hareket ettirme
2- Boruları çizdirme ve borudan geçtikçe skor arttırma
3- Borularla çarpışmayı algılama
4- İlerledikçe yeni borular dizerek sonsuz bir level döngüsünü sağlama ve oyunu hızlandırma
5- Skor başlangıç bitiş tarzı basit UI elementlerini oyuna ekleme

1- Playeri model olarak eklemek istiyorduk fakat kütüphane eklemede yaşadığımız sorunlar ve başka scriptleri eklemek istediğimizde yaşadığımız sorunlar zamanımızın yetmemesine neden oldu. Bu yüzden player'i küre modeli ile gösterdik. Hareketi gerçekçi hale getirmek açısından playerin zıplamadığı süre arttıkça yere düşme hızını arttırdık. Aynı zamanda playerin ekran sınırları içerisinde kalması için kontroller ekledik.

```
void DecreasePlayerYValue(int value)
{
    if(didCollide) return;
    if(isGameStarted)
    {
        timeSinceJump += gravityWithTime * playerSize;
        calculatedY = playerY - (timeSinceJump + gravity) * playerSize;
        playerY = std::max(-4.5,calculatedY);
    }

    glutPostRedisplay();

    didCollide = CollisionControl();

    glutTimerFunc(25, DecreasePlayerYValue, 0);
}
```

2- Borular için glut cylinder fonksiyonunu kullandık. Borular arası boşluğun noktasını belirleyerek üste ve alta 2 boru çizdirip bir bloğu tamamlamış olduk. Player bu silindirlerin orta noktasını geçtiği zaman skor kazanmış olarak kabul edildi.

```
void DrawSinglePipe(double pipeX,double pipeSpaceY)
{
    glColor3f(0,1,0);
    glPushMatrix();
        glTranslated(pipeX,pipeSpaceY + defaultPipeSpacing/2 + defaultPipeHeight,-10);
        glRotated(90,1,0,0);
        glutSolidCylinder(defaultPipeWidth/2,defaultPipeHeight,16,16);
    glPopMatrix();
    glPushMatrix();
        glTranslated(pipeX,pipeSpaceY - defaultPipeSpacing/2 ,-10);
        glRotated(90,1,0,0);
        glutSolidCylinder(defaultPipeWidth/2,defaultPipeHeight,16,16);
    glPopMatrix();
}
```

3- Borularla olan çarpışmayı algılamak için bize çarpabileceğimiz yakınlıkta olan boruları alıp playerin kordinatları bu borulardan herhangi birinin içindeki bir noktayla çarpışıyor mu diye kontrol ettik. Üstteki ve alttaki borular için ayrı kontroller gerektiği için yakındaki boru setini üst ve alt olarak ayırıp kontrol ettik.

```
bool CollisionControl()
{
    for(int i = 0;i<pipeCount;i++)
    {
        if(fabs(pipeXPositions[i] - playerX) > playerSize) continue;

        topPipeVertices[0] = pipeXPositions[i] - defaultPipeWidth / 2;
        topPipeVertices[1] = pipeYPositions[i] + defaultPipeSpacing / 2;

        topPipeVertices[2] = topPipeVertices[0] + defaultPipeWidth;
        topPipeVertices[3] = topPipeVertices[1];

        topPipeVertices[4] = topPipeVertices[0];
        topPipeVertices[5] = topPipeVertices[1] + defaultPipeHeight;

        topPipeVertices[6] = topPipeVertices[1];
        topPipeVertices[7] = topPipeVertices[5];
        //-------------------------------------------------------------//
        botPipeVertices[0] = pipeXPositions[i] - defaultPipeWidth / 2;
        botPipeVertices[1] = pipeYPositions[i] - defaultPipeSpacing / 2;

        botPipeVertices[2] = botPipeVertices[0] + defaultPipeWidth;
        botPipeVertices[3] = botPipeVertices[1];

        botPipeVertices[4] = botPipeVertices[0];
        botPipeVertices[5] = botPipeVertices[1] - defaultPipeHeight;

        botPipeVertices[6] = botPipeVertices[1];
        botPipeVertices[7] = botPipeVertices[5];


        if(playerY + playerSize >= topPipeVertices[1] && playerY - playerSize <= topPipeVertices[5])
        {
            if(playerX + playerSize > topPipeVertices[0] && playerX - playerSize < topPipeVertices[2])
            {
                return true;
            }
        }

        if(playerY - playerSize <= botPipeVertices[1] && playerY + playerSize >= botPipeVertices[5])
        {
            if(playerX + playerSize > botPipeVertices[0] && playerX - playerSize < botPipeVertices[2])
            {
                return true;
            }
        }
    }
```

4- İlerledikçe yeni borular üretmek için oyunlarda optimizasyon amacıyla sıkça kullanılan “Object Pooling” metodunu kullandık. Ekranda maximum 4 tane boru görünebileceği için biz en soldaki borunun ekrandan kaybolmasını bekleyip kaybolur kaybolmaz en sağa ekleyerek toplamda sadece 4 boru ile sonsuz sayıda boruyu simüle edebildik. Bu sayede borular ve bunlar ait değerler bellekte-rendererda gereksiz yer kaplamayarak performansı arttırdı. Ayrıca playeri hareket ettirip kamerayı hareket ettirmek yerine sadece bu 4 tane boruyu X ekseninde sola doğru oyunun hızına göre kaydırdık. Yani aslında player sadece Y de hareket etti. Kamera X ve Y de sabit kaldı ve borular sadece X te hareket etti. Bir sonraki adımda kendisinden bir önceki borunun y pozisyonuna playerin atlayabileceği bir mesafe içerisinde rastgele bir fark ekleyerek oyunu tekrara düşürmekten kaçındık.

```
void MoveThePipes(int value)
{
    if(!isGameStarted || didCollide) return;
    for(int i = 0;i<pipeCount;i++)
    {
        pipeXPositions[i] -= speedFactor;
        if(pipeXPositions[i] < -(pipeCount / 2) * distanceBetweenPipes)
        {
            pipeXPositions[i] += pipeCount * distanceBetweenPipes;
            previousIndex = (i + pipeCount - 1) % pipeCount;
            double nextY = pipeYPositions[previousIndex];
            srand(GetTickCount());
            double randomDifference = (3.0f * rand()) / RAND_MAX - 1.5;
            calculatedPipeY += randomDifference;
            calculatedPipeY = std::max(-3.0,calculatedPipeY);
            calculatedPipeY = std::min(3.0,calculatedPipeY);
            pipeYPositions[i] = calculatedPipeY;
        }

        if(pipeXPositions[i] - playerX > 0 && pipeXPositions[i] - playerX < distanceBetweenPipes)
        {
            if(lastIndex != nextPipeIndex)
            {
                lastIndex = nextPipeIndex;
                currentScore++;
            }
            nextPipeIndex = i;
        }
    }

    glutTimerFunc(10, MoveThePipes, 0);
}
```

5- Playerin oynadığı süre boyunca oyunu oynanabilecek maximum hıza doğru belirli zaman aralıklarıyla yaklaştırdık

```
void IncreaseGameSpeed(int value)
{
    if(speedFactor >= maxSpeedFactor || !isGameStarted || didCollide) return;

    speedFactor += 0.001;
    glutTimerFunc(500,IncreaseGameSpeed,0);
}
```

6- Ekrana yazı bastırmak için glutBitmapCharacter kullandık. Kullanıcıya oyuna başlamak-ölünce yeniden doğmak için ne yapması gerektiğini ekrana yazıp istenilen inputlar geldiğinde istenilen aksiyonları gerçekleştirdik.

```
void drawText(char *string)
{
    char *c;
    glRasterPos3f(-2.0,0.0f, -9.0f);
    for (c=string; *c != '\0'; c++)
    {
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
    }
}
```

Oyuna ses – arka plan textureları ve modeller eklemek istesek de codeblocks ile yaşadığımız sorunlar ve son dönemdeki proje yoğunluğu sebebiyle istediğimiz her şeyi ekleyemedik. Ancak oyunun düzgün ve performanslı bir şekilde çalıştığına inanıyoruz.