Breakout Game Development with Unreal Engine

Title: Implementation of Paddle, Ball, and Brick Mechanics

Author: Martin Revol-Buisson

Introduction

This document explains the implementation of a Breakout-style game using Unreal Engine. The primary goal of the project is to design a playable game with key mechanics such as:

- Paddle movement with boundary restrictions.
- Ball movement and collision with objects.
- Bricks that can be destroyed by the ball.

The work involves coding in C++ within the Unreal Engine environment, focusing on manual collision detection and gameplay logic.

Project Components

1. Paddle

Description:

The paddle allows the player to control the ball's direction by bouncing it off its surface. It moves horizontally and is constrained within the screen boundaries.

Key Features:

- Horizontal movement using `Q` (left) and `D` (right).
- Boundary restrictions to prevent it from leaving the play area.

Code Snippets:

```
void APaddle::Tick(float DeltaTime)
{
    FVector NewLocation = GetActorLocation();
    NewLocation.Y += CurrentMovement * MovementSpeed * DeltaTime;
    NewLocation.Y = FMath::Clamp(NewLocation.Y, LeftBoundary, RightBoundary);
    SetActorLocation(NewLocation);
}
```

---

2. Ball

Description:

The ball moves automatically and bounces off walls, the paddle, and bricks. The collision mechanics are handled manually to allow for precise gameplay control.

Key Features:

- Automatic movement with direction updates upon collision.
- Manual collision detection with paddle and bricks.
- Prevents sticking to the paddle by implementing a collision cooldown.

Code Snippets:

- **Collision with Paddle:**

```
void ABall::CheckPaddleCollision()
{
    if (!PaddleActor || !bCanCollideWithPaddle) return;

    FVector PaddleLocation = PaddleActor->GetActorLocation();
    FVector PaddleScale = PaddleActor->GetActorScale3D();

    float PaddleLeft = PaddleLocation.Y - (200.0f * PaddleScale.Y);
    float PaddleRight = PaddleLocation.Y + (200.0f * PaddleScale.Y);
    float PaddleTop = PaddleLocation.X + (50.0f * PaddleScale.X);

    if (BallLocation.Y >= PaddleLeft && BallLocation.Y <= PaddleRight &&
BallLocation.X >= PaddleTop)
    {
        MovementDirection.X *= -1;
        BallLocation.X = PaddleTop + 1.0f;
        SetActorLocation(BallLocation);
        bCanCollideWithPaddle = false;
        GetWorld()->GetTimerManager().SetTimer(CollisionResetTimer, this,
&ABall::ResetPaddleCollision, 0.2f, false);
    }
}
```

- **Collision with Bricks:**

```
void ABall::CheckBrickCollisions()
{
    TArray<AActor*> FoundBricks;
    UGameplayStatics::GetAllActorsOfClass(GetWorld(), ABrick::StaticClass(),
FoundBricks);

    for (AActor* BrickActor : FoundBricks)
    {
        FVector BrickLocation = BrickActor->GetActorLocation();
```

```
            if (IsCollisionDetected(BallLocation, BrickLocation))
            {
                BrickActor->Destroy();
                MovementDirection.X *= -1;
                break;
            }
        }
    }
```

---

3. Bricks

Description:

Bricks are destructible objects that disappear when hit by the ball.

Key Features:

- Designed as a separate actor class with customizable properties.
- Destroyed upon collision with the ball.

Code Snippets:

```
void ABrick::DestroyBrick()
{
    Destroy();
}
```

Challenges and Solutions

1. Preventing Ball Sticking to Paddle

- **Problem:** Continuous collision detection caused the ball to stick to the paddle.
- **Solution:** Introduced a collision cooldown using a timer.

2. Manual Collision Detection

- **Problem:** Unreal Engine's default collision system was not suitable for the precise control needed.
- **Solution:** Implemented custom bounding box checks for collision logic.

Future Improvements

- **Scoring System:** Add a score counter for every brick destroyed.
- **Multiple Levels:** Introduce progressively harder levels.
- **Brick Types:** Add indestructible or multi-hit bricks.
- **Power-ups:** Include features like paddle size changes or multiple balls.

Conclusion

This project demonstrates the implementation of core gameplay mechanics for a Breakout-style game in Unreal Engine. Through custom coding and manual collision detection, the game achieves precise and responsive controls. Future enhancements will expand on the gameplay experience.