Chapter 3

Player Controllers, Pawns, and Weapons

This chapter will cover player controllers, the player’s pawn, and weapons. A pawn in the UDK represents the physical presence of a player or a computer controlled game character. Think of this as the player’s body in the game. The player controller controls the pawn and translates player input into pawn movement and other actions. Information regarding these topics is presented followed by tutorials. You will find out how to:

- Use the PlayerController and Pawn classes
- Make a pawn visible using a skeletal mesh
- Use camera views
- Set and change your pawns’ views
- Use the Weapon and related classes
- Add weapons to pawns from different views

Player Controller and Pawn Overview

The player input system within the UDK is divided between the player controller and the pawn. The pawn represents the player’s physical presence in the game. When you need to find the player’s location, or rotation you will need to find the location or rotation of the player’s pawn. The player controller takes input from the player, processes this input and translates them into movement for the pawn it controls.

All player controllers must derive from the PlayerController class and all pawns must derive from the Pawn class. For programming on the iOS platform the SimplePC class can be used for the base for custom player controllers and SimplePawn can be used as the base for custom pawns. The advantage of using the SimplePC class as the base of your custom player controller is that this class already has set up joystick and touch input.
zones as well as some other features like footstep sounds and simulated player breathing through camera motion. The advantage of using the SimplePawn is that it features “head bobbing” which makes the camera move up and down as the player walks around. This makes the player’s view more realistic. Head bobbing is done by setting the bScriptTickSpecial=true which is the default value in SimplePawn. The actual head bobbing is done in the TickSpecial() function in the SimplePawn class. To turn head bobbing off, set bScriptTickSpecial=false in a derived pawn class. The main update function for the PlayerController class is PlayerTick(), which is called at regular intervals, or “ticked” once per frame update of the game.

Here you can add in custom code to handle items unique to your particular game. For example, if a goal in your game is to retrieve a certain item then you can check for the player’s possessions for this item in PlayerTick(). The way you would do this is override the PlayerTick() function in your custom player controller, execute the normal PlayerTick() function through the Super prefix and then execute your custom code afterwards. See Listing 3–1.

**Listing 3–1. Customizing the PlayerTick() function**

```c++
function PlayerTick(float DeltaTime)
{
    Super.PlayerTick(DeltaTime);
    // Add in additional code here for your custom player controller
    If (Jazz3Pawn(Pawn).Lives <= 0)
    {
        Gameover = true;
    }
}
```

The modified PlayerTick() function checks if the player’s has more lives and if not then sets the Gameover variable to true.

The PlayerMove() function is called every time PlayerTick() is called and calculates new acceleration and rotation values for the player. This function then calls ProcessMove() which then executes the actual move. It is in the ProcessMove() function that we can change how the player’s pawn responds to the player’s movement input. For example, in your custom controller class if you declared a function ProcessMove() that is defined in the PlayerWalking state then this new function will override the default ProcessMove() function.

```c++
state PlayerWalking
{
    ignores SeePlayer, HearNoise, Bump;
    function ProcessMove(float DeltaTime, vector NewAccel, eDoubleClickDir DoubleClickMove, rotator DeltaRot)
    {
        // Place custom code for player movement here to override the default
    }
}
```

The UpdateRotation() function is responsible for updating the controller’s rotation and the pawn’s rotation.