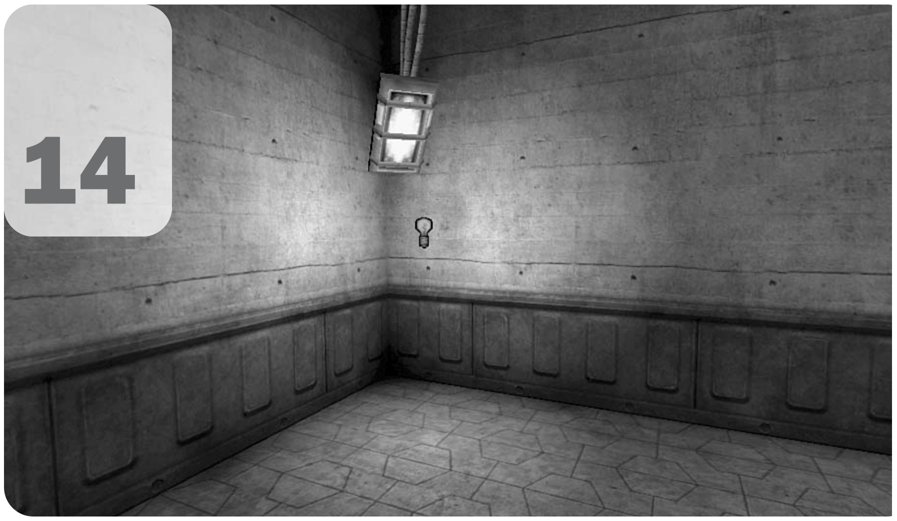
Chapter 14



Creating Scripted Sequences

In many games, you find that the player character is not the only element involved with the environment. Often the game drives many actions and events to enhance the overall experience. For example, you might be playing a game in which you're moving through a science laboratory and see scientists going about their duties, assistants moving equipment, secretaries carrying important items, and so forth. Within Unreal, these elements must be added into a level as scripted sequences. This chapter covers how you can use the ScriptedSequence Actor to create a variety of custom events within your Unreal levels.

Adding interactive or reactive game elements, making non-player characters (NPCs) perform actions, and other such events add an incredible amount of gameplay value to your levels. For example, say your character is behind a closed door, and you want a scientist NPC on the other side to open the door for you. With scripted sequences, you can tell a bot, or NPC, to move to the button that opens the door, start a door-pressing animation, and then open the door.

To create these scripted sequences, you have the ScriptedSequence Actor, located under Keypoint > AIScript in the Actor Class browser. This Actor contains all the main functions for controlling NPCs, or xPawns, and controlling how events are processed. Subsets of this Actor are the ScriptedTrigger and UnrealScriptedSequence Actors. The ScriptedTrigger should be used when handling events, as when you used it in **CHAPTER 9**, "Interactive Elements," to create an advanced elevator. UnrealScriptedSequence Actors are used when you want to control the behavior of bots, as you did in **CHAPTER 12**, "Advanced Bot/AI Navigation."

This chapter starts with a general discussion of how to create Actions, the backbone of scripted sequences. From there, you learn how to control the timing of events and see that certain Actions are latent, meaning they must be finished before moving on, and some are not. Then you learn how to use logical conditions to control whether certain Actions should occur.

Using an Actions List

Scripted sequences have an `Actions` property located under their AIScript category. This property can hold a series of Actions that are performed from the top to the bottom of the list. This series of Actions is known as the *Actions list*. An Action can perform a variety of tasks: trigger an event, play a sound, spawn an Actor, control artifical intelligence (AI), and much more. When a level starts, the first Action is performed, followed by the second, and continuing until reaching the final Action in the list. Unless there's an Action that alters the sequence, such as an `Action_GOTOACTION`, the ScriptedSequence carries out the last Action and then stops.

Using Latent and Non-latent Actions

Actions are categorized into two main types, depending on how they are performed: latent and non-latent. *Latent* actions cause Actions in the list to pause until the current Action is finished before they're carried out. *Non-latent* actions cause some effect or event and then immediately proceed to the next Action without waiting until the first Action is completed. For example, if you use an `Action_PLAYANIM` to play an animation of a character waving, and then immediately follow it with an `Action_PLAYSOUND`, both Actions would seem to happen simultaneously.

TABLES 14.1 and **14.2** describe the available non-latent and latent actions.

TABLE 14.1 Non-latent Actions

Action	Description	Notes
`ACTION_ASSetPlayerSpawnArea`	Enables or disables `PlayerSpawnManagers`.	Used in Assault gametype
`ACTION_ASTeleportToSpawnArea`	Causes players to spawn at points controlled by the specified `PlayerSpawnManager`.	Used in Assault gametype

17 6922 ch14 10/12/04 11:12 AM Page 3

TABLE 14.1 Continued

Action	Description	Notes
`ACTION_ChangeLevel`	Loads another map. Useful for creating Matinee sequences that span multiple maps.	
`Action_ChangeObjectiveTeam`	Updates the specified objective so that it belongs to the specified team.	
`ACTION_ChangeScript`	Leaves the current script and starts running the specified script.	AI control only
`ACTION_ChangeWeapon`	Causes the controlled pawn to switch to the specified weapon. The pawn must already have the weapon in its inventory.	AI control only
`Action_ConsoleCommand`	Performs a console command.	
`Action_CROUCH`	Causes the controlled pawn to crouch until an `ACTION_Run` or `ACTION_Walk` is used.	AI control only
`ACTION_DamageActor`	Causes all Actors with the specified `Tag` to receive damage of a certain type and amount.	
`ACTION_DamageInstigator`	Damages the player or NPC that caused the current script to run.	
`ACTION_DestroyActor`	Removes all Actors of the specified `Tag` from the level.	
`ACTION_DestroyPawn`	Removes the controlled pawn from the level.	AI control only
`ACTION_DisableObjective`	Allows an objective to be bypassed and counted as complete.	Used in Assault game-type
`ACTION_DisableThisScript`	When performed, causes the in-game (bot) AI to ignore this script. Applies only to UnrealScriptedSequence Actors.	
`ACTION_DisplayMessage`	Displays or broadcasts a message to one or all players in the game.	

TABLE 14.1 Continued

Action	Description	Notes
`ACTION_EndSection`	Used to mark the end of a group of Actions controlled by an `Action_IFCONDITION` or `Action_IFRANDOMPCT`.	
`ACTION_FireWeapon`	Causes the controlled pawn to start or stop firing its current weapon.	AI control only
`ACTION_FireWeaponAt`	Causes the controlled pawn to shoot at an Actor, designated by a `Tag`.	AI control only
`Action_FORCEMOVETOPOINT`	Snaps the controlled pawn to the specified point.	AI control only
`ACTION_FreezeOnAnimEnd`	Freezes the controlled pawn's animation and physical movement after its current animation finishes.	AI control only
`Action_GOTOACTION`	Jumps the script to the specified Action. Useful for making scripted sequences repeatable.	
`ACTION_GotoMenu`	Displays the game's main menu. A custom menu class can be specified with the `MenuName` property.	
`Action_IFCONDITION`	If the associated `TriggeredCondition`'s `bEnabled` property is False, it causes the script to skip to the Action after the next `Action_ENDSECTION`.	
`Action_IFRANDOMPCT`	Based on the given probability, the script may or may not j ump to the Action after the next `Action_ENDSECTION`.	
`ACTION_Jump`	Causes the controlled pawn to jump. The type of jump can be controlled with the `JumpAction` property.	AI control only
`Action_KILLINSTIGATOR`	Kills the player that caused the script to run. Should be used only with ScriptedTriggers.	

TABLE 14.1 Continued

Action	Description	Notes
`Action_LEAVESEQUENCE`	Exits the current script and ignores any further Actions.	
`ACTION_LocalizedMessage`	Broadcasts a localized message.	
`Action_PLAYAMBIENTSOUND`	Causes the ScriptedSequence's `AmbientSound` properties to be set according to this action.	
`Action_PLAYANIM`	Causes the controlled pawn to play an animation.	AI control only
`Action_PLAYANNOUNCEMENT`	Plays an announcement sound in the game's currently selected announcer voice. The specified sound must be a valid announcer sound.	
`ACTION_PlayerViewShake`	Causes the instigator's view to shake as though hit by a weapon. Occurs only when the instigator is within a specified radius from the ScriptedSequence Actor.	
`Action_PLAYLOCALSOUND`	Causes all players to hear a sound. This sound is not location based.	
`Action_PLAYMUSIC`	Sets the current music to the specified song. This Action can set the music for just the instigator or for all players in the level. The song must be a file from the Music directory under the folder where you installed Unreal, without the `.ogg` extension. For example, if you want to play `KR-DM1.ogg`, you would set the `Song` property to `KR-DM1`.	
`Action_PLAYSOUND`	Plays a sound that originates from the ScriptedSequence Actor.	
`Action_RUN`	Causes the controlled pawn to run when moving. (Disables the effect of `Action_CROUCH` and `Action_WALK`.)	AI control only

TABLE 14.1 Continued

Action	Description	Notes
`Action_SETALERTNESS`	Sets the alertness of the controlled pawn.	AI control only
`ACTION_SetHidden`	Hides all Actors with the specified `Tag`.	
`ACTION_SetObjectiveActiveStatus`	Determines whether the specified objective is active.	Used in Assault gametype
`ACTION_SetPhysics`	Sets the instigator's current physics type.	
`Action_SETVIEWTARGET`	Sets the location that the controlled pawn tries to look toward. Because this Action is non-latent, you need more time to pass after this Action to see the effect.	AI control only
`ACTION_ShootTarget`	Causes the controlled pawn to shoot at its current target.	AI control only
`ACTION_SpawnActor`	Spawns an Actor of the specified class. The Actor can be spawned from the location of the ScriptedSequence Actor or from a controlled pawn's location. Additional location and rotation offset can also be set.	
`Action_STOPANIMATION`	Causes the controlled pawn to stop playing its current animation.	AI control only
`ACTION_StopShooting`	Aborts the effect of `ACTION_ShootTarget`.	AI control only
`ACTION_SubTitles`	Causes the heads-up display's (HUD's) current subtitles to advance.	Used in assault
`ACTION_ThrowWeapon`	Causes the controlled pawn to throw its current weapon in the direction specified by the `WeaponVelocity` property.	AI control only

17 6922 ch14 10/12/04 11:12 AM Page 7

TABLE 14.1 Continued

Action	Description	Notes
`Action_TRIGGEREVENT`	Causes the specified event to be broadcast.	
`Action_USE`	Causes the controlled pawn to use the player.	
`Action_WALK`	Causes the controlled pawn to walk when moving.	AI control only

TABLE 14.2 Latent Actions

Action	Description	Notes
`Action_DrawHUDMaterial`	Renders the specified material as an overlay onto the HUD. You can specify the width, height, and size the material is displayed at. Also, you can set the amount of time the material is displayed.	
`ACTION_FinishRotation`	Waits until the instigator or controlled pawn is facing its goal.	
`ACTION_Freeze`	Freezes the controlled pawn's animation and physical movement.	AI control only
`Action_MOVETOPLAYER`	Causes the controlled pawn to navigate to the instigating player. Script running continues when the pawn reaches the player.	AI control only
`Action_MOVETOPOINT`	Causes the controlled pawn to navigate to the specified point. Script running continues when the pawn reaches that point.	AI control only
`ACTION_PlayExplosionSound`	Plays a random explosion sound. With the `SoundEmitterActorTag` property, you can specify a location for the sound to be emitted from. (Default location is the scripted sequence itself.)	
`Action_TELEPORTTOPOINT`	Teleports the instigator to the specified point. The audiovisual teleportation effect can be enabled or disabled.	

TABLE 14.2 Continued

Action	Description	Notes
`Action_TURNTOWARDPLAYER`	Causes the controlled pawn to face the player.	AI control only
`Action_WAITFORANIMEND`	Causes script running to pause until the controlled pawn's current animation finishes.	AI control only
`Action_WAITFOREVENT`	Causes script running to pause until the specified event is broadcast.	
`Action_WAITFORPLAYER`	Causes script running to pause until the player is within the specified distance of the controlled pawn.	AI control only
`Action_WAITFORTIMER`	Causes script running to pause for the time specified. Note that the time is in seconds.	

TUTORIAL 14.1 shows how to create a simple ScriptedTrigger that uses both latent and non-latent Actions to open a door in a complex manner.

TUTORIAL 14.1: Using a ScriptedTrigger

1. Open `Tutorial14_01_Start.ut2`. In this map, you'll create a UseTrigger that plays a sound, opens a door, and turns on a "caution" light.
2. In the Actor Class browser, navigate to Triggers, and click UseTrigger.
3. Add the UseTrigger in front of the door (see **FIGURE 14.1**). To make the UseTrigger visible in-game, open the Properties window for the UseTrigger, go to the Advanced category, and set the `bHidden` property to False.

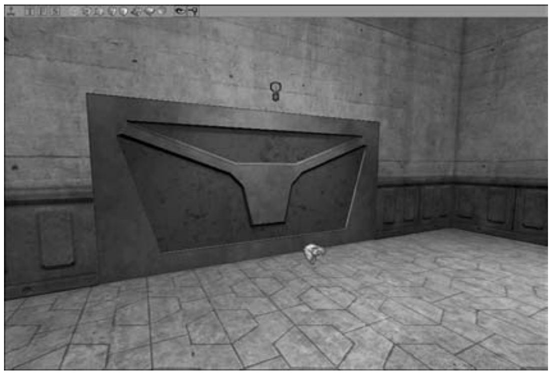


FIGURE 14.1 The UseTrigger placed in front of the door.

4. Because you have three separate Actions to perform, this is a great place for a ScriptedTrigger. Open the Actor Class browser, expand Keypoint > AIScript > ScriptedSequence, and click ScriptedTrigger.

5. Place the ScriptedTrigger next to the UseTrigger (see **FIGURE 14.2**). Although the location of a ScriptedTrigger is irrelevant, place it near the UseTrigger because the two triggers will be working with one another.

6. Open the Properties window for the UseTrigger. Under the Events category, type `MultiTrigger` as the setting for the `Event` property. This event will be the one that triggers the chain of events.

7. Open the Properties window for the ScriptedTrigger. Under the AIScript category, select the `Actions` property, and click the Add button. By default, the Actions in the Actions list begin evaluating as soon as the level starts. In this case, however, you don't want anything to happen until you activate the UseTrigger. This means you need to wait for the `MultiTrigger` event to be triggered before any Actions take place. To do this, select `Action_WAITFOREVENT` and click the New button. Set the `ExternalEvent` property to `MultiTrigger` (see **FIGURE 14.3**). Until this event is triggered, the next Action is not performed.

8. Now you need to add three Actions to the list: one to play the sound, one to open the door, and one to turn on the light. Select the `Actions` property and click the Add button three times. For index [1], select `Action_PLAYSOUND` and click New. Open the Sound browser. If you're using Unreal Tournament 2004, open the `IndoorAmbience` package and select `door10`. If you're using the Unreal Runtime Engine, open the `EM_Runtime_A` package and select `door12`. Back in the Properties window for `Action_PLAYSOUND`, select the `Sound` property, and click the Use button.

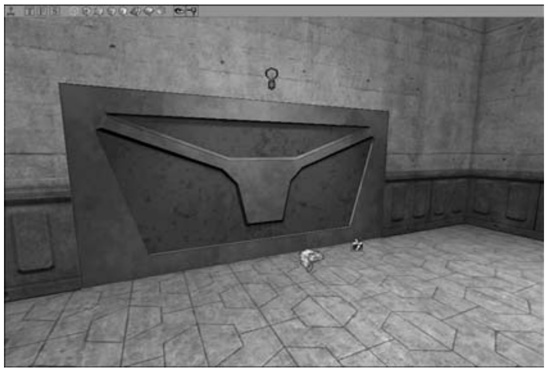
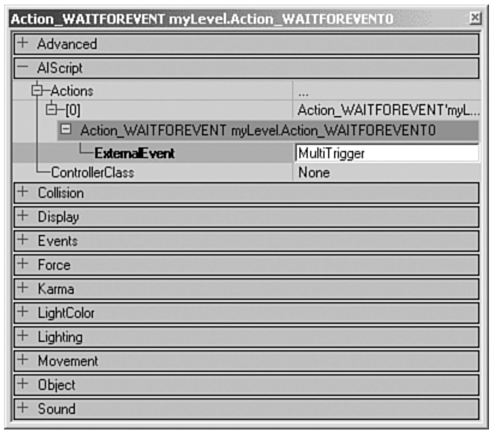


FIGURE 14.2 The ScriptedTrigger placed next to the UseTrigger.


```
Action_WAITFOREVENT myLevel.Action_WAITFOREVENT0
+ Advanced
− AIScript
  Actions                                   ...
    [0]                                     Action_WAITFOREVENT'myL...
      Action_WAITFOREVENT myLevel.Action_WAITFOREVENT0
        ExternalEvent                       MultiTrigger
  ControllerClass                           None
+ Collision
+ Display
+ Events
+ Force
+ Karma
+ LightColor
+ Lighting
+ Movement
+ Object
+ Sound
```
FIGURE 14.3 One Action created for the ScriptedTrigger.

NOTE

You need to manually enter `MultiTrigger` for the `Event` property.

9. For index [2], select `Action_TRIGGEREVENT` and click New. Because the `Tag` property for the door is `BlastDoor`, set the `Event` property of the `Action_TRIGGEREVENT` to `BlastDoor`.
10. For index [3], select `Action_TRIGGEREVENT` and click New. The `Tag` property for the light is `CautionLight`, so set the `Event` property of the `Action_TRIGGEREVENT` to `CautionLight`. This setting activates a dynamic light that's already in the scene.
11. Save the map, and give it a test spin. Note that the door opens, the sounds plays, and the light switches on after you hit the trigger. Also, notice that although the Actions are taking place in a specific order, they all appear to be happening at the same time.

END TUTORIAL 14.1

Controlling xPawns

Now that you have an idea of how to control events, you're ready to take it one step further and control actual non-player characters (NPCs). This brings another property of ScriptedSequences into the spotlight: `ControllerClass`. By default, it's set to None. For controlling an xPawn, you must set it to `ScriptedController`. Basically, this setting overrides the `ControllerClass` for the xPawn, allowing the ScriptedSequence Actor to take control of the xPawn and alter its behavior. If you didn't use this setting, it would default to a bot-style AI, which would, in effect, spawn an additional player into the game.

To associate the xPawn with the ScriptedSequence Actor, set the xPawn's `AIScriptTag` property, found under the AI category, to the `Tag` property of the ScriptedSequence. **TUTORIAL 14.2** demonstrates how to set up a simple scenario in which a ScriptedSequence controls an xPawn.

TUTORIAL 14.2: Controlling a Pawn

1. Open `Tutorial14_02_Start.ut2`. In this tutorial, you're going to create an NPC that runs to the player, beckons for the player to follow, and then opens the door. You already have a series of PathNodes set up for the NPC to follow; you just need to give the NPC the proper commands. For more information on creating bot paths, please refer to **CHAPTER 12**, "Advanced Bot/AI Navigation."
2. First, you need to create the NPC. Open the Actor Class browser, navigate to Pawn > UnrealPawn, and click xPawn. In the Top view, move the xPawn to the lower right of the level, near the PathNode. Rotate the xPawn to point toward the PathNode (see **FIGURE 14.4**).
3. Open the Actor Class browser, navigate to Keypoint > AIScript, and click ScriptedSequence Actor. Place this Actor next to the xPawn.
4. In the Properties window for the ScriptedSequence Actor, go to the Events category and change the `Tag` property to `PawnController`. Also, under the AIScript Category, set the `ControllerClass` property to `ScriptedController`.

5. Now open the Properties window for the xPawn and go to the AI category. Set the `AIScriptTag` to `PawnController`, which forces the xPawn to listen for behavior commands from the ScriptedSequence Actor. After doing this, you'll see a line connecting the ScriptedSequence icon to the xPawn (see **FIGURE 14.5**).
6. Now that everything is set up properly, you can create the Actions for the xPawn to follow. Open the Properties window for the ScriptedSequence. First, you need to wait for the player to hit the UseTrigger, which sends the `CallNPC` event, so add an `Action_WAITFOREVENT` and set the `ExternalEvent` to `CallNPC`.
7. Second, the xPawn needs to move to the player. To do this, add an `Action_MOVETOPOINT` and set the `DestinationTag` to `CallPoint`, which is the `Tag` property of the PathNode next to the UseTrigger. Because you have PathNodes leading to the `DestinationTag`, the xPawn moves correctly from PathNode to PathNode until reaching the destination.

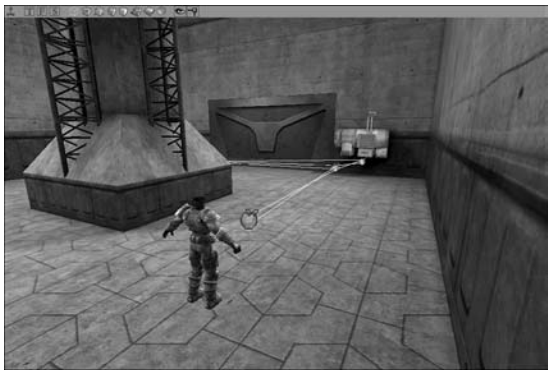


FIGURE 14.4 xPawn ready for action.

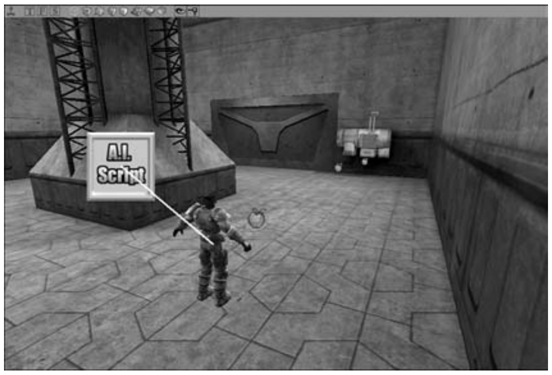


FIGURE 14.5 ScriptedSequence connected to the xPawn.

8. At this point, you need the xPawn to look at the player. To do this, add an `Action_TURNTOWARDPLAYER`.
9. With the xPawn now facing the character, you must make sure the xPawn is stationary and not playing any animations. For this, add an `Action_WAITFORTIMER` and set the `PauseTime` to 0.5. Now add an `Action_PLAYANIM` and set the `BaseAnim` to `gesture_beckon`. Because `Action_PLAYANIM` is non-latent, you need to make sure the animation ends before moving to the next Action; therefore, add an `Action_WAITFORANIMEND` and leave `Channel` set to 0.
10. To make the xPawn turn toward the console as he moves toward it, add an `Action_SETVIEWTARGET` and change the `ViewTargetTag` property to `ConsoleTag`. As before, add an `Action_MOVETOPOINT` and set the `DestinationTag` to `ConsolePoint`, which is the `Tag` property of the PathNode in front of the console.

11. As before, you need to ensure that the xPawn is stationary and isn't playing any other animations. So add an `Action_WAITFORTIMER` and set the `PauseTime` to 0.2, which is lower than before because the previous `Action_SETVIEWTARGET` should already have the xPawn facing in the right direction. Now add an `Action_PLAYANIM` and set the `BaseAnim` to `gesture_halt`. To ensure that the animation ends, add an `Action_WAITFORANIMEND`.
12. Finally, you need the door to open. Add an `Action_TRIGGEREVENT` and set `Event` to `BlastDoor`, the `Tag` property of the door Mover.
13. Save the map and run it. When you're in-game, go over to the UseTrigger and press the Use key (the E key, by default). If it doesn't work correctly, use **TABLE 14.3** as a reference for the Actions in this tutorial.

TABLE 14.3 Actions for the ScriptedSequence

Index	Action	Properties to Set
[0]	`Action_WAITFOREVENT`	`ExternalEvent = CallNPC`
[1]	`Action_MOVETOPOINT`	`DestinationTag = CallPoint`
[2]	`Action_TURNTOWARDPLAYER`	N/A
[3]	`Action_WAITFORTIMER`	`PauseTime = 0.5`
[4]	`Action_PLAYANIM`	`BaseAnim = gesture_beckon`
[5]	`Action_WAITFORANIMEND`	N/A
[6]	`Action_SETVIEWTARGET`	`ViewTargetTag = ConsoleTarget`
[7]	`Action_MOVETOPOINT`	`DestinationTag = ConsolePoint`
[8]	`Action_WAITFORTIMER`	`PauseTime = 0.2`
[9]	`Action_PLAYANIM`	`BaseAnim = gesture_halt`
[10]	`Action_WAITFORANIMEND`	N/A
[11]	`Action_TRIGGEREVENT`	`Event = BlastDoor`

END TUTORIAL 14.2

You can also use ScriptedSequences when you're creating intro movies with Matinee and custom animations. **TUTORIAL 14.3** shows you how to use a custom character with custom animation to create a movie sequence. For more information on using Matinee, see **CHAPTER 13**, "Matinee: Creating Custom Cinematics."

TUTORIAL 14.3: Using Matinee with ScriptedSequences

1. Open `Tutorial14_03_Start.ut2`. In this level, you have a Matinee sequence where the camera flies around the level and circles around the inside of a room. Inside this room, you're going to place a custom character and cause it to play some of the custom animation included with it.

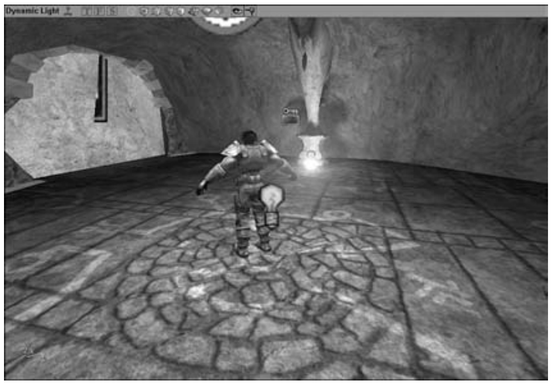


FIGURE 14.6 xPawn placed in the level.

2. To start off, place an xPawn in the center of the room, which is located in the center of the level (see **FIGURE 14.6**). Change `ControllerClass` to `ScriptedController`. Also, under the UnrealPawn category, set the `bNoDefaultInventory` property to `True` so that the character isn't holding a gun.

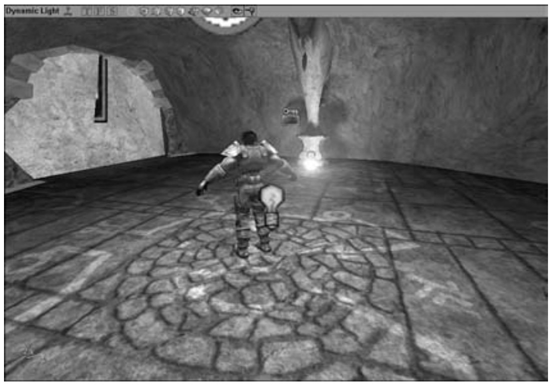


FIGURE 14.7 Custom character rotated and moved into position.

3. Because you want to use your own custom character, not the default, open the Animation browser so that you can load the correct character. Open the `Chapter14_Anim.ukx` package. With this character visible in the browser, open the Properties window for the xPawn. Under the Display category, select the `Mesh` property and click Use. Also, select the `Skins` property and click Empty. Rotate the xPawn so that it points between the two interpolation points. Also, move the xPawn so that it rests on the floor (see **FIGURE 14.7**). Because you need to control this pawn with a ScriptedSequence, go to the AI category and set the `AIScriptTag` property to `LFController`.

4. Next, create a ScriptedSequence near the xPawn. Under the Events category, change the `Tag` property to `LFController`. If you ran the level now, you would notice that the xPawn is in the wrong pose. Playing an animation at this point would cause the character to snap from this pose to the first frame of the animation you want. To correct this, you have a single-frame animation (KickStart) that leaves the character at the correct pose for starting the animation. From there, you need to start the Matinee sequence, which can be triggered with the `StartMovie` event. **TABLE 14.4** shows the Actions needed for the ScriptedSequence Actor.

TABLE 14.4 Actions for the ScriptedSequence

Index	Action	Properties to Set
[0]	`Action_PLAYANIM`	`BaseAnim = KickStart`
[1]	`Action_TRIGGEREVENT`	`Event = StartMovie`

5. Because you want the animation to start when the camera enters the room, you need to have the Matinee sequence trigger an event. To do this, open the Matinee dialog box. Select the StartMovie scene and switch to the Actions tab. Then select the second Action from the bottom. Now switch to the Sub Actions tab, and select the third Sub Action from the top. Add a SubActionTrigger. Under the Trigger category, set the `EventName` property to `StartAction`.
6. With that set up, reopen the `ScriptedSequence` property and add the final Actions that will wait for the event, as shown in **TABLE 14.5**, and then play the animation. At the very end, quit to the main menu.

TABLE 14.5 Actions for the ScriptedSequence

Index	Action	Properties to Set
[2]	`Action_WAITFOREVENT`	`ExternalEvent = StartAction`
[3]	`Action_PLAYANIM`	`BaseAnim = Kick`
[4]	`Action_WAITFORANIMEND`	N/A
[5]	`Action_GotoMenu`	N/A

7. That's it! Save and run the map. The character's animation won't begin until the Matinee sequence tells it to. **FIGURE 14.8** shows a screenshot of the custom animation.

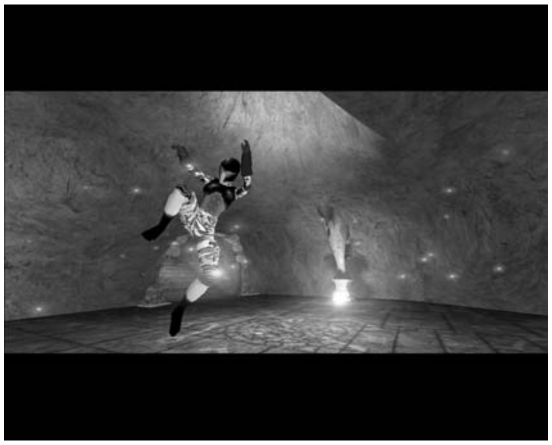


FIGURE 14.8 The character playing the custom animation.

END TUTORIAL 14.3

Logical Conditions

Although the simple Actions you've seen so far can handle most situations you run into, at times you want the result or running of a script to differ depending on external conditions. To this end, you have the following three Actions (described previously in **TABLE 14.1**): `Action_IFCONDITION`, `Action_ENDSECTION`, and `Action_IFRANDOMPCT`. To use the `Action_IFCONDITION`, you must also have a TriggeredCondition Actor, which is located under the Triggers section of the Actor Class browser. When this Actor is triggered, an internal value toggles between True and False. To use it, you create a section of Actions that are blocked off by an `Action_IFCONDITION` and `Action_ENDSECTION`. If the internal value is True, the Actions within the block are carried out, and those outside it are disregarded.

TUTORIAL 14.4: Using If Conditions

1. Open `Tutorial14_04_Start.ut2`. In this tutorial, you create a system in which a pawn must activate a control unit before the player can open the door.
2. First, you'll set up the ScriptedSequence to make the pawn run over to a PathNode and send the `IC_DoorControl` event to the `TriggeredCondition`. Add a ScriptedSequence Actor next to the existing Trigger. In the Properties window for the ScriptedSequence, first set the `ControllerClass` to `ScriptedController`. Next, the xPawn must be associated with the ScriptedSequence. The xPawn's `AIScriptTag` is currently set to `NPCScript`, so set the ScriptedSequence's `Tag` property to `NPCScript`. Now add the Actions shown in **TABLE 14.6** to the ScriptedSequence.

TABLE 14.6 Actions for the ScriptedSequence

Index	Action	Properties to Set
[0]	`Action_WAITFOREVENT`	`ExternalEvent = CallBot`
[1]	`Action_WALK`	N/A
[2]	`Action_SETVIEWTARGET`	`ViewTargetTag=BotPanel`
[3]	`Action_MOVETOPOINT`	`DestinationTag= SwitchPoint`
[4]	`Action_WAITFORTIMER`	`PauseTime=0.2`
[5]	`Action_PLAYANIM`	`BaseAnim=gesture_halt`
[6]	`Action_WAITFORANIMEND`	N/A
[7]	`Action_TRIGGEREVENT`	`Event=IC_DoorControl`

3. Now you need to actually create the TriggeredCondition. In the Actor Class browser, go the Triggers section and click TriggeredCondition. Place this Actor next to the trigger.

4. In the Properties window for the TriggeredCondition, go to the TriggeredCondition category, and set `bTriggerControlled` to True so that the TriggeredCondition can be externally controlled by a trigger. Also, set the `Tag` property under the Events category to `IC_DoorControl`.
5. The UseTrigger in front of the second PathNode sends the `TestConsole` event. Therefore, open the ScriptedTrigger Properties window and create the set of Actions shown in **TABLE 14.7**.

TABLE 14.7 Actions for the ScriptedTrigger

Index	Action	Properties to Set
[0]	`Action_WAITFOREVENT`	`ExternalEvent = TestConsole`
[1]	`Action_IFCONDITION`	`TriggeredConditionTag = IC_DoorControl`
[2]	`Action_PLAYSOUND`	`Sound = MenuSounds.J_MouseOver`
[3]	`Action_TRIGGEREVENT`	`Event= OpenDoor`
[4]	`Action_ENDSECTION`	`N/A`
[5]	`Action_GOTOACTION`	`ActionNumber= 0`

As you can see, what's between `Action_IFCONDITION` and `Action_ENDSECTION` isn't performed unless the TriggeredCondition is toggled to True by the ScriptedSequence.

6. Save the map and test the level. Run over to the trigger and watch the xPawn run to the first PathNode. Now you should be able to open the door.

END TUTORIAL 14.4

`Action_IFRANDOMPCT` is used when you want some group of Actions to happen randomly. As with `Action_IFCONDITION`, you must also have an `Action_ENDSECTION` to signify the end of the block. The one property available in this Action is `Probability`. Setting this property to 1 causes the Actions inside the block to always be performed. A value of 0, however, means the Actions are never carried out. Finally, setting it to 0.5 means there's a 50% chance of the Actions being performed.

TUTORIAL 14.5 shows how to create a flickering dynamic light by using `Action_IFRANDOMPCT`. This method gives you more control over how the light flickers and is an excellent demonstration of how useful this Action can be.

TUTORIAL 14.5: Using Random Conditions

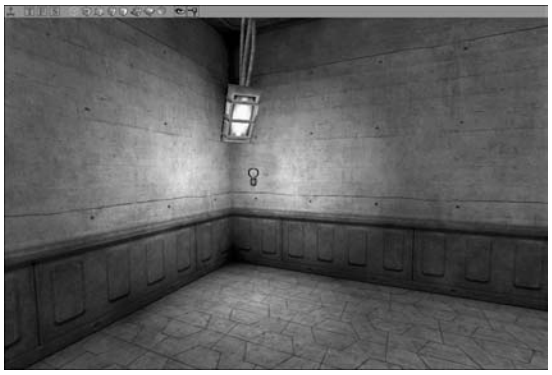


FIGURE 14.9 TriggerLight placed next to the static mesh.

1. Open `Tutorial14_05_Start.ut2`. This level is nothing but a simple cube with a light fixture where you'll be placing the flickering light.
2. In the Actor Class browser, select Light > TriggerLight. Place this light next to the static mesh (see **FIGURE 4.9**). In the Properties window for the TriggerLight, go to the LightColor category and set `LightBrightness` to 200, `LightHue` to 150, and `LightSaturation` to 150. Also, under the Lighting category, set `LightRadius` to 10. Under the Events category, change the `Tag` property to `FlickerLight`. Finally, under the Object category, set `InitialState` to `TriggerToggle` so that the light responds to external events.
3. Now add a ScriptedTrigger near the TriggerLight, and then add the Actions shown in **TABLE 14.8**.

TABLE 14.8 Actions for the ScriptedTrigger

Index	Action	Properties to Set
[0]	`Action_WAITFORTIMER`	`PauseTime = 0.10`
[1]	`Action_IFRANDOMPCT`	`Probability = 0.50`
[2]	`Action_TRIGGEREVENT`	`Event = FlickerLight`
[3]	`Action_ENDSECTION`	N/A
[4]	`Action_GOTOACTION`	`ActionNumber= 0`

When the level starts, you wait for 0.1 seconds, and then a random condition says that there's a 50% chance you switch the light on or off. After that, you just jump back to the top, wait another 0.1 seconds, and so on.

4. Save the map and test the level. You should now have a flickering light.

END TUTORIAL 14.5

Summary

In this chapter, you've learned everything you need to create more interactive levels and intro movies. You started with a look at how to use scripted sequences, and then learned how to use Actions to control what takes place in your sequences, along with a list of the primary Actions available and the differences between latent and non-latent Actions. Finally, you learned how to use logical conditions with your scripted sequences to create conditional actions. With these simple foundations, you should be able to create a massive variety of scripted sequences to enhance the look and feel of your levels.