Panel Transition Sequence Manager Documentation

Overview

The `PanelTransitionSequenceManager` script is responsible for managing the transitions between various UI panels in a Unity project. The panels correspond to different stages of the game sequence, such as instructions, different rounds, and end screens like "Game Over" or "You Win." This documentation explains the functionality and usage of the script, intended for integration into a group project.

UI Panels

The script controls the following UI panels:

- **laserWelcomePanel**: The initial welcome panel.
- **reverseWelcomePanel**: A reverse animation of the welcome panel.
- **instructionsPanel**: The panel showing game instructions.
- **reverseInstructionsPanel**: A reverse animation of the instructions panel.
- **laserActivatedPanel**: The panel indicating laser activation.
- **reverseLaserActivatedPanel**: A reverse animation of the laser-activated panel.
- **roundOnePanel**: The panel for round one.
- **roundTwoPanel**: The panel for round two.
- **roundThreePanel**: The panel for round three.
- **gameOverPanel**: The panel displayed when the game is over.
- **youWinPanel**: The panel displayed when the player wins.

Test Mode

The script supports a test mode that allows developers to manually trigger panel transitions using keyboard keys. The keys are configurable and are used to simulate the different stages of the game.

Key Functionality

1. **Initialization**:
    - The script deactivates all panels at the start and activates the `laserWelcomePanel`.
2. **Panel Transitions**:
    - Transitions between panels are managed based on game events or keyboard input in test mode.
    - Custom transitions allow moving from a non-reverse panel to its reverse counterpart.
3. **Handling Input**:
    - In test mode, specific keys can activate corresponding panels.

    - When awaiting input, the script can advance the panel sequence based on key presses.
4. **Custom Transitions**:
    - The script supports custom transitions, particularly from non-reverse panels to their reverse counterparts and from the reverse laser-activated panel to the rounds.
5. **Sequential Panel Play**:
    - The script can play a sequence of panels with a specified delay between transitions.
    - After the sequence, the game can transition to the first round.
6. **Round Indicators**:
    - The script includes functionality to play animations for round indicators based on the current round number.

Integration Steps

1. **Add the Script to a GameObject**:
    - Attach the `PanelTransitionSequenceManager` script to an appropriate GameObject in your Unity scene.
2. **Assign Panels in the Inspector**:
    - In the Unity Inspector, assign the corresponding UI panel GameObjects to the script's public fields.
3. **Configure Test Mode Keys**:
    - If using test mode, configure the key bindings in the Inspector for various transitions.
4. **Activate and Test**:
    - Ensure the script is active and run the game. Use the configured keys to test transitions in test mode.

Animator Integration

The script assumes the use of an Animator to handle panel animations. Below are screenshots demonstrating the Animator setup for one of the panels:

Animator Setup:

- **Entry State**: The starting state of the animation sequence.
- **LaserWelcome**: The animation state for the laser welcome panel.
- **RightLeftLaserInto**: The subsequent animation state triggered after the welcome panel.

Ensure your Animator is correctly configured to match the panel transitions defined in the script.

GameManager.cs

Overview

`GameManager` handles the overall game flow, including wave management, enemy spawning, laser control, and game outcome determination. It interacts with `PanelTransitionSequenceManager` to display UI panel transitions based on the current wave and game state.

Fields

- **Managers**
    - `FindSpawnPositions _findSpawnPositions`: Manages spawn positions for enemies and other game objects.
    - `PassthroughManager passthroughManager`: Manages passthrough layer settings.
    - `EnemySpawnManager _enemySpawnManager`: Handles enemy spawning logic.
    - `AlienPortalManager _alienPortalManager`: Manages alien portal operations.
    - `Hovl_DemoLasers leftEyeLaser`: Manages the left eye laser.
    - `Hovl_DemoLasers rightEyeLaser`: Manages the right eye laser.
    - `PanelTransitionSequenceManager panelTransitionManager`: Manages UI panel transitions.
- **Wave Settings**
    - `_wave1Enemies`: Number of enemies in Wave 1.
    - `_wave2Enemies`: Number of enemies in Wave 2.
    - `_wave3Enemies`: Number of enemies in Wave 3.
- **Timing Settings (in seconds)**
    - `gameStartDelay`: Delay before game starts.
    - `triggerAreaSpawnDelay`: Delay before spawning the trigger area.
    - `powerUpPassthroughDelay`: Delay before activating passthrough after power-up.
    - `waveStartDelay`: Delay before starting each wave.

    - `_wave1Time`: Duration of Wave 1.
    - `_wave2Time`: Duration of Wave 2.
    - `_wave3Time`: Duration of Wave 3.
- **Audio**
    - `AudioSource backgroundMusicSource`: Audio source for background music.
    - `AudioSource enemyHitSoundSource`: Audio source for enemy hit sound.
    - `AudioClip enemyTakingDamage_AudioClip`: Audio clip for enemy taking damage.
    - `GameObject enemyDestroyEffect`: Effect to play when an enemy is destroyed.
- **Events**
    - `UnityEvent onIntroStart`: Event triggered at the start of the intro.
    - `UnityEvent onIntroCompleted`: Event triggered at the end of the intro.
    - `UnityEvent onGameStarted`: Event triggered when the game starts.
    - `UnityEvent onTriggerAreaSpawned`: Event triggered when the trigger area is spawned.
    - `UnityEvent onPowerUpSpawn`: Event triggered when a power-up is spawned.
    - `UnityEvent<CurrentWave> onNewWaveDisplay`: Event triggered to display the new wave.
    - `UnityEvent<CurrentWave> onNewWaveStart`: Event triggered when a new wave starts.
    - `UnityEvent<CurrentWave> onCurrentWaveEnd`: Event triggered when the current wave ends.
    - `UnityEvent onGameOver`: Event triggered when the game is over.
    - `UnityEvent onGameWon`: Event triggered when the game is won.
    - `UnityEvent onEnemyHit`: Event triggered when an enemy is hit.
    - `UnityEvent<GameObject> onEnemyDestroyed`: Event triggered when an enemy is destroyed.

Methods

- **Initialization**
    - `Start()`: Initializes game variables and events, starts the alien portal spawning.
    - `ResetGameVariables()`: Resets game variables to their initial state.
    - `ResetWaveVariables()`: Resets wave-specific variables to their initial state.
    - `InitializeEvents()`: Sets up the event listeners for the game manager.
    - `IEnumerator SpawnAlienPortal()`: Spawns the alien portal after a delay.
- **Game Start**
    - `OnIntroCompleted()`: Starts the game after the intro is completed.

    - `IEnumerator StartGameCoroutine()`: Delays the game start by a specified duration.
    - `HandleGameStarted()`: Handles the logic when the game starts.
- **Trigger Area**
    - `SpawnTriggerArea()`: Spawns the trigger area in the current room.
    - `OnUserEnteredTriggerArea(GameObject gameObj)`: Handles the logic when the user enters the trigger area.
    - `OnUserEnteredTriggerArea()`: Invokes the power-up spawn event.
    - `HandlePowerUpSpawn()`: Handles the logic when a power-up is spawned.
    - `IEnumerator StartWave_AfterDelay()`: Delays the start of the next wave.
- **Waves**
    - `IEnumerator StartNextWave()`: Starts the next wave and manages enemy spawning.
    - `CurrentWaveEnded()`: Handles the logic when the current wave ends.
    - `SetNextWave()`: Sets the next wave.
- **Lasers**
    - `StartShootingLaser()`: Starts shooting lasers from the eyes.
    - `StopShootingLaser()`: Stops shooting lasers from the eyes.
- **Enemies**
    - `HandleEnemyHit()`: Handles the logic when an enemy is hit.
    - `EnemySpawned(GameObject enemy)`: Adds a spawned enemy to the list.
    - `HandleEnemyDestroyed(GameObject enemyObj)`: Handles the logic when an enemy is destroyed.
- **Game Outcome**
    - `GameOutcome FindGameOutcome(bool timeIsUp)`: Determines the game outcome based on remaining enemies and time.
    - `HandleGameOver()`: Handles the logic when the game is over.
    - `HandleGameWon()`: Handles the logic when the game is won.
- **Utilities**
    - `Update()`: Updates the game timer and checks for wave completion.
    - `FormatDisplayTime(float seconds)`: Formats the display time for the remaining wave duration.
    - `Log(string logText)`: Logs messages with the class name for debugging purposes.
