GOLF COURSE NPC BALL

COLLECTOR

-

Ercan Acar

2024

INDEX

DESCRIPTION

Develop a simple 3D game set in a golf course environment with a forest backdrop. Create an NPC that walks to gather randomly distributed golf balls on the map while returning to a stable golf cart for scoring.

Overview

This documentation outlines the key mechanics governing NPC behavior in the game, particularly focusing on the collection and deposition of golf balls by NPCs. The NPCs interact with golf balls and a golf cart, with specific conditions dictating their actions.

Game Mechanics

1. NPC Movement and Navigation

- **Nav Mesh**: NPCs use Unity's Nav Mesh system for navigation, allowing them to move through the game world while avoiding obstacles.
- **Movement Triggers**: NPCs react to specific triggers (colliders) in the game environment to initiate movement towards golf balls or the golf cart.

2. Ball Collection

- **Trigger Detection**: NPCs detect golf balls using trigger colliders. Upon entering the trigger area of a golf ball, they can initiate the gathering process.
- **Animation**: When an NPC collects a golf ball, a "gather" animation is triggered, simulating the action of picking up the ball.
- **State Management**: Once an NPC has collected a ball, its state is updated to indicate that it is now carrying the ball.

3. Ball Deposition

- **Golf Cart Interaction**: NPCs detect when they enter the trigger area of the golf cart. If carrying a ball, they can initiate the deposition process.
- **Scoring**: Upon depositing a ball, the points associated with that ball are added to the player's score.

- **Resetting State**: After depositing a ball, the NPC searches for the next available golf ball to collect.

4. Decision-Making Algorithm

The NPC's decision-making is based on health status, ball proximity, and ball level:

- **Health Check**:
    - If the NPC's health is above 50%, it prefers to collect higher-level balls.
    - If health is 50% or lower, the NPC prioritizes the nearest ball.
- **Ball Evaluation**:
    - The NPC evaluates all active golf balls in the game.
    - It calculates the distance to each ball and assesses their levels.
    - The decision-making process involves comparing the distances and levels of the available golf balls to determine which one to pursue.
- **Movement Decision**: Once the best ball is identified, the NPC's destination is set to the ball's position, and it begins moving towards it.

5. Game Over Condition

- **Game Over Trigger**: When certain conditions are met (not specified in this documentation), a GameOver() method is called, stopping NPC movement and triggering the game manager's game-over logic.

Assumptions

1. **Game Manager Integration**: The NPC Controller class assumes the existence of a singleton Game Manager that oversees global game state, including health management and scoring.

2. **Proper Tagging**: Golf balls and golf carts are expected to be properly tagged within the Unity environment. The class relies on these tags to identify objects during trigger events.

3. **Nav Mesh Setup**: A valid Nav Mesh must be created in the game environment to facilitate NPC navigation. This ensures that NPCs can move effectively around the game world.

4. **Animation Controller**: The NPC Controller assumes that an Animator component is attached to the NPC, with defined parameters for triggering animations related to gathering and depositing balls.

5. **Health Threshold**: The decision-making algorithm uses a fixed health threshold (50) to determine whether the NPC should prefer higher-level balls. This threshold may be adjusted based on game design but should be consistent to maintain balance.

6. **Golf Ball Component**: Each golf ball must have a Golf Ball component attached, which provides information about the ball's level and points.

7. **Active Golf Balls**: The game must maintain a list of active golf balls that the NPCs can interact with, which is managed by the Game Manager.

Overview

The Game Manager class is responsible for managing the overall game state and facilitating interactions between various subsystems within the game. It utilizes the Singleton design pattern to ensure that only one instance exists throughout the game's lifecycle, making it globally accessible to other components.

Key Responsibilities

- Manage references to various game managers, including:
    - Ball Manager
    - Points Manager
    - Health Manager
    - Menu Controller
    - Audio Manager
- Manage the initialization of the NPC Controller based on the player Game Object tagged as "Player".
- Ensure that the Game Manager persists across scene loads.

Game Mechanics

Singleton Pattern

The Game Manager implements the Singleton pattern, ensuring that only one instance exists throughout the game. This is achieved through:

- A static property Instance that provides global access to the Game Manager.

- An Awake method that checks if an instance already exists. If not, it assigns itself as the singleton instance; if it does, it destroys the new instance to maintain the singleton nature.

Game Manager References

The Game Manager holds serialized references to various other managers:

- **Ball Manager**: Controls the behavior and mechanics of the game ball.
- **Points Manager**: Manages the scoring system and keeps track of player points.
- **Health Manager**: Tracks the player's health and handles health-related mechanics.
- **Menu Controller**: Manages UI menus and transitions between different game states.
- **Audio Manager**: Responsible for playing sound effects and background music.

NPC Controller Initialization

The Game Manager looks for a Game Object tagged as "Player" during the Awake method and retrieves its NPC Controller component if found. This allows the game to interact with the NPC associated with the player character, enabling various gameplay mechanics related to the NPC.

Assumptions

1. **Player Game Object**:
    - It is assumed that there is always a Game Object tagged as "Player" present in the scene. If no player is found, the NPC Controller will remain unassigned, which may lead to issues in gameplay if other scripts depend on it.

2. **Serialized References**:

    - The game managers (Ball Manager, Points Manager, Health Manager, etc.) must be properly assigned to the Unity Inspector for the Game Manager to function correctly. If any reference is missing, it may result in Null Reference Exception errors during gameplay.

3. **Scene Management**:

    - The Don't Destroy on Load method is used to keep the Game Manager instance alive across scene transitions. It assumes that maintaining state across scenes is necessary for the game's design.

4. **NPC Controller Dependency**:

    - Other components in the game may rely on the NPC Controller being assigned to function correctly. If the Game Manager cannot find the player, those components may encounter issues.

Overview

The Ball Manager class is responsible for managing the spawning and lifecycle of golf ball objects in the game. It generates a specified number of golf balls within defined boundaries, assigns levels to the balls based on their distance from a specified NPC starting position, and provides functionality to destroy the spawned balls when needed.

Key Responsibilities

- Spawn a specified number of golf balls at random positions within given boundaries.
- Assign difficulty levels to each ball based on their distance from a designated NPC start position.
- Provide methods to retrieve active spawned balls and to destroy all spawned balls.

Level Assignment

Each spawned ball is assigned a difficulty level based on its distance from the NPC starting position:

- **Level 1**: Assigned if the distance is less than 30 units (Easy level).
- **Level 2**: Assigned if the distance is less than 55 units but greater than or equal to 30 units (Medium level).
- **Level 3**: Assigned if the distance is 55 units or more (Hard level).

Ball Lifecycle Management

- **Destroy Balls**: The Destroy Balls method iterates through the list of spawned balls, destroying each one and clearing the list afterward to remove references to the destroyed objects.

Data Structures

- **Spawned Balls List**: A private list _spawnedBalls stores all the spawned golf balls, allowing for easy management of their lifecycle and access to their states.

Assumptions

1. **Prefab Existence**:
    - The golfBallPrefab must be assigned to the Unity Inspector for the Ball Manager to function correctly. If not assigned, the game will throw a Null Reference Exception when attempting to instantiate the golf balls.
2. **Boundary Setup**:
    - It is assumed that the boundary Game Objects (boundary1 and boundary2) are placed correctly in the scene. Their positions determine the spawning area for the golf balls.
3. **NPC Start Position**:
    - The script assumes the existence of a defined npcStartPosition, which is crucial for calculating the distance to determine the level of the spawned balls.
4. **Active State of Balls**:
    - The script does not manage the active state of the golf balls beyond spawning and destroying them. If the balls are deactivated elsewhere in the code, the Get Active Spawned Balls method will reflect that state.
5. **Memory Management**:
    - The Destroy Balls method clears the _spawnedBalls list after destroying the balls, assuming that this practice is essential for proper memory management and avoiding potential memory leaks.

Overview

The Points Manager class is responsible for managing the points system in the game. It handles adding points, displaying the current score in the UI, and animating point increases to enhance the player's experience.

Key Responsibilities

- Maintain a total points count and provide methods to manipulate and display points.
- Animate the increase in points to create a visually appealing experience for the player.
- Reset points and update the UI accordingly.

Algorithms

1. **Add Points**:
    - In the Add Points method:
        - The starting points are stored in startPoints.
        - The _totalPoints variable is incremented by the provided points.
        - The sound effect for gaining points is played using GameManager.Instance.AudioManager.PlaySfx("Gain").
        - The coroutine Animate Points Increase is initiated to animate the points increase.
2. **Animate Points Increase**:
    - The Animate Points Increase coroutine performs the following steps:

- Initializes elapsedTime to zero.
- Continuously update the elapsedTime until it reaches the animationDuration.
- Within the loop:
    - Calculates the interpolation factor t based on elapsedTime relative to animationDuration.
    - Uses Mathf.Lerp to interpolate between startPoints and endPoints, rounding to the nearest integer to determine currentPoints.
    - Updates the pointsText to show the current points.
    - Yields until the next frame for smooth animation.
- After the loop, it ensures that pointsText displays the final point total.

Assumptions

1. **UI Text Reference**:
    - The pointsText must be assigned to the Unity Inspector for the Points Manager to function correctly. If not assigned, a Null Reference Exception will occur when trying to update the text.
2. **Audio Manager Existence**:
    - It is assumed that the Audio Manager is correctly set up within the Game Manager and that it has a method named Play Sfx that takes a string argument for the sound effect name.

3. **Animation Duration**:

    - The animationDuration variable is set in the Inspector. A reasonable value should be assigned to ensure a smooth animation experience; otherwise, the animation might be too fast or slow.

4. **Point Values**:

    - The method assumes that the point values passed to Add Points are positive integers. Negative values should be handled elsewhere in the code to avoid unintended behavior.

Overview

The Health Manager class is responsible for managing the player's health in the game. It tracks the current health value, updates a health bar UI element, and decreases health over time if certain conditions are met.

Key Responsibilities

- Maintain the player's health value and provide methods to manipulate and access health.
- Update a UI health bar to reflect the current health state.
- Manage health depletion and trigger game-over conditions when health reaches zero.

Game Mechanics

Health Management

- **Initial Health**: The player starts with a maximum health value of 100, represented by the health variable.
- **Health Bar**: An Image UI element (the health bar) visually represents the player's current health. The fill amount of this image is updated based on the current health value.
- **Health Decrease Rate**: The variable healthDecreaseRate determines how quickly the health decreases over time.

Resetting Health

- The Reset method restores the player's health to its maximum value (100) and updates the health bar to full (normalized value of 1).

Health Update Logic

- The Update method decreases health continuously over time while the NPC is active (i.e., when isStopped is false). The health value is clamped to ensure it does not go below zero or exceed 100.

Game Over Condition

- If health reaches zero, the NPC's actions are halted, a debug message is logged indicating exhaustion, and a Game Over method is called on the NPC Controller to manage the end-of-game logic.

Algorithms

1. **Reset Health**:
    - The Reset method is called to set health back to 100 and update the health bar to a full state.
2. **Update Health**:
    - In the Update method:
        - It checks if the NPC is not stopped using GameManager.Instance.NPCController.isStopped.
        - If the NPC is active, the health is decremented based on healthDecreaseRate and Time.deltaTime, ensuring a consistent decrease over time.
        - The health value is clamped between 0 and 100 to prevent invalid values.
        - The health bar's fill amount is updated to reflect the current health as a normalized value between 0 and 1.
        - If health drops to zero, the following actions are taken:

- A debug message "NPC is exhausted!" is logged to indicate that the NPC has no remaining health.
- The Game Over method of the NPC Controller is called to handle the game-over scenario.

3. **Get Current Health**:
    - The Get Health method returns the current health value, allowing other classes to access this information if needed.

Assumptions

1. **Health Bar Reference**:
    - The healthBar must be assigned in the Unity Inspector for the Health Manager to function correctly. If it is not assigned, a Null Reference Exception will occur when trying to update the health bar.
2. **NPC Controller Existence**:
    - It is assumed that NPC Controller is properly instantiated and accessible via GameManager.Instance.NPCController.
3. **Health Decrease Rate**:
    - The healthDecreaseRate variable is adjustable in the Inspector. A reasonable value should be assigned to ensure that health decreases at a desired rate.
4. **Time.deltaTime Usage**:
    - The use of Time.deltaTime assumes that the Update method will be called once per frame, which is standard in Unity.

Overview

The Menu Controller class manages the user interface transitions between different game states, including the main menu, in-game interface, and game-over screen. It manages user interactions and audio management, ensuring a smooth experience for the player.

Key Responsibilities

- Control the visibility of the main menu, in-game interface, and game-over screen.
- Manage audio playback for different game states.
- Handle buttons click events to transition between menus and initiate gameplay.

Algorithms

1. **Menu Initialization**:
    - The Start method initializes the menu state by playing the main menu music and setting the visibility of the menu screens.
2. **Play Button Logic**:
    - In OnPlayButtonClicked, the following occurs:
        - Sound effects are played for feedback.
        - The game state is prepared by spawning balls and stopping NPC actions.
        - The UI is updated to show the in-game interface.
        - The DelayedStart coroutine is started to manage the transition to gameplay after a brief pause.

3. **Exit Logic**:
    - The OnExitButtonClicked method handles application quitting based on the context (editor vs. standalone).
4. **Menu Navigation**:
    - The OnMenuButtonClicked method resets the game state and navigates back to the main menu, ensuring a clean slate for a new game session.
5. **Game Over Logic**:
    - In the GameOver method, the game state is updated to reflect the end of gameplay, stopping music and displaying the game-over screen.

Assumptions

1. **AudioManager Existence**:
    - The AudioManager instance in GameManager must be properly instantiated and configured to handle music and sound effects.
2. **GameManager Initialization**:
    - It is assumed that GameManager and its components (such as BallManager and NPCController) are correctly set up before the MenuController is invoked.
3. **GameObject References**:
    - All serialized GameObject references (inGame, mainMenu, gameOver) must be assigned in the Unity Inspector to avoid NullReferenceException.
4. **Sound Effect Availability**:
    - The specified sound effects (e.g., "Click", "Gain", "Nice", "Game Over") must be available and correctly configured in the AudioManager.

Overview

The Audio Manager class is responsible for managing audio playback in the game, including background music and sound effects. It facilitates smooth audio transitions between different game states and allows for dynamic audio control through user interactions.

Key Responsibilities

- Play and stop background music for different game states (main menu and in-game).
- Manage sound effects and control their playback.
- Manage volume settings for both music and sound effects.

Algorithms

1. **Audio Source Initialization**:
    - In the Awake method, audio sources are created, and volume settings are applied to ensure proper audio configuration when the game starts.
2. **Sound Effect Access**:
    - The Initialize Sfx Dictionary method constructs a dictionary that allows efficient access to sound effects based on names, promoting a clean and organized structure for managing audio clips.
3. **Playing Audio**:
    - When invoking methods to play music or sound effects, the class checks for the validity of audio clips to prevent errors and maintain a robust user experience.

4. **Volume Management**:

    - The class provides methods for adjusting music and sound effects volume, allowing developers to implement settings menus or other volume controls dynamically.

Assumptions

1. **Audio Clip Assignments**:

    - The audio clips (mainMenuMusic, inGameMusic, and sound effects) must be correctly assigned in the Unity Inspector to avoid Null Reference Exception.

2. **Sound Effect Class Structure**:

    - The Sound Effect class is expected to have properties for both name and clip, which are used to store and access audio clips effectively.

3. **Game Object Context**:

    - The Audio Manager should be attached to an active Game Object in the scene to ensure audio sources are properly instantiated.

4. **Performance Considerations**:

    - The use of Play One Shot for sound effects is assumed to be suitable for the gameplay experience, providing instantaneous feedback without interrupting the main audio source.

Overview

The Sound Effect class serves as a container for audio clips and their associated names, allowing for easy management of sound effects within the game. It is particularly useful in the context of audio management systems like the Audio Manager, where a list of sound effects can be maintained and accessed efficiently.

Key Responsibilities

- Store the name and audio clip of a sound effect.
- Enable easy serialization and management of sound effects within the Unity Inspector.

Properties

1. name

- **Type**: string
- **Description**: The name assigned to the sound effect. This name is used to identify and access the audio clip associated with the sound effect. It should be unique within the context of sound effects to prevent ambiguity.

2. clip

- **Type**: Audio Clip
- **Description**: The audio clip that represents the sound effect. This clip contains the actual audio data that will be played when the sound effect is triggered.

SOURCE

https://sketchfab.com/3d-models/18th-hole-old-course-st-andrews-links-6b699b69b30b4e169aa911c78cd80bd1

https://sketchfab.com/3d-models/low-poly-golf-flag-animated-a6ebddcfff9a4c8a912551530ebfdb45

https://sketchfab.com/3d-models/golf-ball-c7c9bd93d57d454e90af30bff5ee5877

https://assetstore.unity.com/packages/2d/gui/icons/sleek-essential-ui-pack-170650

https://www.canva.com/

https://www.mixamo.com/

https://soundimage.org/looping-music/

https://www.myinstants.com/en/instant/mouse-click-by-ek6-9658/

https://www.myinstants.com/en/instant/minecraft-level-up-sound/

https://www.myinstants.com/en/instant/-click-nice/

https://pixabay.com/sound-effects/search/game-over/

https://www.flaticon.com/free-icon/golf-ball_6141041?term=golf&page=2&position=23&origin=search&related_id=6141041