# Golf Collector Project Documentation

Eren Çakar

October 17, 2024

## Contents



# 1 Introduction

This document provides an overview and detailed documentation of the Golf Collector project. The project is a Unity-based game where golf balls are spawned, initialized, and collected by a singular NPC. This documentation was written with the intention of informing developers who wish to extend the project about the reasons, caveats, and results of implementation decisions.

# 2 Class Descriptions

## 2.1 AbstractGolfBallInitializer

The ‘AbstractGolfBallInitializer‘ is an abstract class that defines the interface for initializing golf balls. It contains an abstract method ‘Initialize‘ which must be implemented by any subclass. This design allows for different initialization strategies to be easily swapped in and out.

**Methods:**

- **Initialize(GameObject golfBall)**: This method initializes the data values of the given golf ball, such as setting its point-level.

**Extensibility:** To extend the initialization logic, create a new class that inherits from ‘AbstractGolfBallInitializer‘ and implement the ‘Initialize‘ method.

## 2.2 GolfBall

The ‘GolfBall‘ class represents an individual golf ball in the game. It is responsible for managing its appearance based on its difficulty level. The class allows dynamic changes to the ball's material and provides visual cues in the editor.

**Fields:**

- **materials (List<Material>)**: A list of materials used for different difficulty levels of the golf ball.
- **level (GolfBallLevel)**: The current difficulty/point level of the golf ball, affecting its appearance.

**Methods:**

- **SetLevel(GolfBallLevel level)**: This method sets the difficulty level of the golf ball and updates its material accordingly.
- **GetLevel()**: This method returns the current difficulty level of the golf ball.
- **OnDrawGizmos()**: This method visually represents the golf ball in the Unity editor with colored spheres based on the current level.

**Implementation Concepts:** The class uses a ‘MeshRenderer‘ to change the material of the golf ball dynamically. It also employs the ‘OnDrawGizmos()‘ method to provide visual feedback in the Unity editor, enhancing the development experience.

**Configurability:** You can assign different materials in the Unity inspector for each difficulty level, allowing for flexible customization of the golf ball's appearance.

## 2.3 GolfBallSpawner

The ‘GolfBallSpawner‘ class is responsible for spawning golf balls in the game world. It uses a prefab for the golf balls and an initializer to set their initial state. The number of golf balls to spawn is configurable.

**Fields:**

- **golfBallPrefab (GameObject)**: The prefab used for initiating the golf ball game objects.
- **initializer (AbstractGolfBallInitializer)**: The golf ball initializer script used for each golf ball object right after initiating them.
- **numberOfGolfBalls (int)**: The number of total golf balls to spawn.

**Methods:**

- **SpawnGolfBalls()**: This method gets the NavMesh of the current scene and spawns golf balls at random positions on walkable areas of the NavMesh.

**Implementation Concepts:** The class uses a coroutine to handle the spawning process, allowing for asynchronous operations.

**Configurability:** You can assign any desired ‘AbstractGolfBallInitializer‘ subclass to this class's respective member variable to achieve desired independent initialization of the spawned golf balls.

## 2.4 AbstractGolfBallSensory

The AbstractGolfBallSensory is an abstract class designed to manage the detection and tracking of golf balls in the game. It provides the framework for sensory systems that can detect, blacklist, or whitelist golf balls. Any subclass inheriting from this class must implement the defined abstract methods, ensuring flexibility for different sensory strategies.

**Methods:**

- **GetGolfBalls():** This method returns an array of all detected golf balls in the game world.
- **BlackList(GameObject golfBall):** Adds the specified golf ball to a blacklist, preventing certain interactions or detection.

- **IsBlackListed(GameObject golfBall):** Checks if a specific golf ball is on the blacklist, returning a boolean value.
- **WhiteList(GameObject golfBall):** Removes the specified golf ball from the blacklist, allowing regular interactions.

**Extensibility:** To extend the sensory logic, create a subclass that inherits from AbstractGolfBallSensory and implement each of the abstract methods. This allows for custom detection strategies, such as limiting the detection to specific areas or adding unique conditions for blacklisting/whitelisting golf balls.

## 2.5 AbstractGolfCollectorDecisionMaker

The AbstractGolfCollectorDecisionMaker is an abstract class responsible for implementing decision-making logic for the golf ball collector NPC. It provides a framework for determining which golf balls to collect and how to make decisions on the collector's behavior. Subclasses are required to implement custom logic for decision-making.

**Methods:**

- **DecideFor(GolfBallCollector collector):** This method is responsible for setting up the golf collector whom should be taken as a reference to decide on the best golf ball to pick up.
- **GetBestGolfBall(GameObject[] golfBalls):** This method returns the best golf ball from the provided array of GameObject instances. The criteria for selecting the "best" ball should be implemented in subclasses.

**Extensibility:** To customize the decision-making logic, create a subclass that inherits from AbstractGolfCollectorDecisionMaker. Implement the DecideFor method to define the specific actions the collector should take, and implement the GetBestGolfBall method to provide criteria for selecting the most suitable golf ball to collect.

## 2.6 Billboard

The Billboard class is responsible for ensuring that a GameObject (such as a UI element or a 3D model) always faces the camera. This is commonly used for creating in-game elements like health bars or indicators that need to remain readable regardless of the camera's position.

**Important Implementation Details:**

- **mainCameraTransform:** Stores a reference to the main camera's Transform component.
- **Start():** Initializes the camera reference by fetching the main camera's transform at the start of the game.
- **LateUpdate():** Updates the GameObject's rotation every frame so that it constantly faces the camera. It uses the LookAt method to align the GameObject towards the camera's forward direction.

**Use Cases:** The Billboard class is particularly useful for creating dynamic UI elements in 3D space that must be oriented towards the player at all times, ensuring clarity and visibility.

## 2.7 CameraController

The CameraController class manages the movement and rotation of the camera in the game. It allows for smooth control of the camera, including rotation based on mouse input and movement using keyboard controls. The camera's behavior is also affected by the game's pause state.

**Fields:**

- **speed (float):** Controls the movement speed of the camera. The speed is doubled when the Left Shift key is held.
- **sensitivity (float):** Determines how sensitive the camera is to mouse movements for rotation.
- **maxYAngle (float):** Limits the vertical camera rotation to prevent it from flipping over. The value is clamped between -maxYAngle and maxYAngle.

**Methods:**

- **Start():** Subscribes to the OnPauseEvent from the GameManager to manage camera behavior when the game is paused or resumed.
- **Update():** Handles both camera rotation and movement. Rotation is based on mouse input, and movement is based on keyboard input (WASD keys). The camera's speed is increased when holding Left Shift.
- **OnPause(bool isPaused):** Pauses or resumes the camera by locking or unlocking the mouse cursor when the game is paused or resumed.
- **LockMouse():** Locks the mouse cursor to the center of the screen and hides it during gameplay.
- **UnlockMouse():** Unlocks the mouse cursor and makes it visible when the game is paused.

**Important Implementation Details:**

- The camera's rotation is updated based on mouse input, and its movement is controlled by keyboard input (WASD) with a speed boost when the Left Shift key is pressed.
- The OnPause method ensures the camera control is disabled when the game is paused, and the mouse is unlocked for interaction.

**Use Cases:** This class is useful for controlling a free-moving camera in a 3D environment, providing first-person or spectator-style movement and camera control.

## 2.8 GameManager

The GameManager class is a singleton responsible for managing the game's state, including pausing, resuming, restarting, and quitting the game. It provides an easy way to control game flow and globally accessible pause state management. Other scripts can subscribe to the OnPauseEvent to be notified when the game is paused or resumed.

**Fields and Properties:**

- **Instance (static):** A static reference to the singleton instance of the GameManager. Ensures only one instance exists at runtime.
- **IsPaused (bool):** A read-only property that indicates whether the game is currently paused.

**Events:**

- **OnPauseEvent:** A delegate event that other classes can subscribe to in order to be notified when the game is paused or resumed. It passes a boolean value indicating the current pause state.

**Methods:**

- **Awake():** Ensures the GameManager follows the singleton pattern. It initializes the Instance and pauses the game when first loaded.
- **PauseGame():** Pauses the game by setting the Time.timeScale to 0, freezing all in-game actions, and invokes the pause event.
- **ResumeGame():** Resumes the game by setting Time.timeScale to 1, allowing normal game flow to continue, and triggers the pause event with false.
- **RestartGame():** Restarts the current scene by reloading it. Resets the IsPaused flag to false.
- **QuitGame():** Exits the game by calling Application.Quit(). In editor mode, this will not have an effect, but it will work in a built version of the game.
- **SetIsPaused(bool isPaused):** Sets the internal IsPaused property and triggers the OnPauseEvent if there are any subscribers.

**Important Implementation Details:**

- The class ensures that the game is paused when first loaded and provides mechanisms to pause or resume the game as needed.
- By using Time.timeScale, the entire game can be paused without affecting other time-independent systems (like UI or menus).
- Other game objects can subscribe to the OnPauseEvent to react to game state changes, such as locking controls or stopping certain animations.

**Use Cases:** The GameManager class is essential for controlling the game's overall state, providing global pause functionality, and facilitating clean, centralized control over game flow actions such as pausing, restarting, and quitting the game.

## 2.9 GolfBallCollector

The GolfBallCollector class controls the behavior of an NPC responsible for collecting golf balls on a 3D golf course map. It uses a combination of AI decision-making and pathfinding to choose and collect the most valuable golf balls while managing health and score. The collector can also return to a golf cart to store collected balls when their carrying capacity is full.

**Key Components:**

- **NavMeshAgent:** The class uses Unity's NavMeshAgent to navigate the 3D space and move toward target golf balls or the golf cart.
- **Health System:** The collector has a health system that depletes over time, which may cause the agent to stop moving if health drops to zero.
- **Golf Ball Collection:** The NPC collects golf balls based on the value and availability, using the GolfBallSensory and DecisionMaker components to decide which ball to pursue.
- **Score System:** When the collector returns to the golf cart with golf balls, their score increases based on the value of the collected balls.

**Fields:**

- **maxGolfBalls:** Maximum number of golf balls the collector can carry at one time.
- **golfBallSensory:** An abstract class that provides information about the golf balls in the environment.
- **decisionMaker:** An abstract class that decides the best golf ball to collect.
- **golfCart:** A reference to the position of the golf cart where balls are deposited.
- **startHealth, damagePerSecond:** Parameters for health management, where health decreases over time.
- **abandonAfterSeconds:** Time after which a pursued golf ball is abandoned if unreachable.

**Methods:**

- **Start():** Initializes the NPC's health and score, subscribes to health and score UI updates, and starts the decision-making process.
- **Update():** Handles the health reduction, ball abandonment, decision-making to pursue a new ball, or return to the cart when fully loaded.
- **SetCurrentGolfBall():** Sets the current target golf ball and calculates how long it will take to reach it.
- **BlackList() / WhiteList():** Manages golf balls that are no longer worth pursuing due to abandonment or inefficiency.
- **CanGoForABetterBall():** Determines if there's a better ball available for collection based on distance and value.

- **GoFor():** Directs the NPC to move toward the selected golf ball.
- **ReturnToGolfCart():** Directs the NPC to move toward the golf cart to stash collected balls.
- **TakeGolfBall():** Collects or swaps the current golf ball for a more valuable one if the NPC is carrying too many balls.
- **StashGolfBalls():** Deposits all collected balls into the golf cart and increases the score accordingly.
- **OnTriggerEnter():** Handles the interaction when the NPC reaches a ball or the golf cart.
- **OnTriggerExit():** Handles rechecking blacklisted balls to potentially collect them if the condition changes.

**Additional Notes:**

- The NPC uses pathfinding to navigate to golf balls and return to the cart.
- Golf balls are "timed," meaning they will be abandoned if not reached within a certain time.
- The class supports swapping golf balls if a more valuable one is found, ensuring the NPC optimizes its collection.

This structure allows for flexible AI behavior with extendable sensory and decision-making components, making the NPC capable of strategic collection tasks in a dynamic environment.

## 2.10 Health

The Health class manages the health system for game entities, providing functionality to adjust health values while ensuring that they remain within valid bounds. It also includes event notifications to inform other systems of health changes.

**Fields:**

- **maxHealth (float):** The maximum health value for the entity, which cannot be less than zero and is assigned during initialization.
- **currentHealth (float):** The current health of the entity, which can change during gameplay.

**Events:**

- **OnHealthChanged:** An event that notifies subscribers whenever the health changes, providing the new current health and the maximum health as parameters.

**Methods:**

- **Health(float maxHealth):** Constructor that initializes the health system with a specified maximum health value and sets the current health to this maximum.
- **IncreaseHealth(float amount):** Increases the current health by a specified amount, ensuring it does not exceed the maximum health. It also triggers the OnHealthChanged event.
- **DecreaseHealth(float amount):** Decreases the current health by a specified amount, ensuring it does not drop below zero. This method also triggers the OnHealthChanged event.
- **GetMaxHealth():** Returns the maximum health value.
- **GetCurrentHealth():** Returns the current health value.

**Important Implementation Details:**

- Health modifications are clamped to ensure that current health remains within valid bounds (0 to maxHealth).
- The OnHealthChanged event allows other components to react to health changes, such as updating health bars or triggering animations.

**Use Cases:** The Health class is essential for managing the health of characters or entities in a game. It can be utilized for player characters, NPCs, or any objects requiring health management, enabling responsive gameplay based on health status.

## 2.11 OmnipotentGolfBallSensory

The OmnipotentGolfBallSensory class extends the AbstractGolfBallSensory class and provides functionality for managing golf balls in the game. It allows for the detection of golf balls through tagging and provides methods to manage the state of each golf ball as either blacklisted or whitelisted.

**Fields:**

- **golfBallTag (static string):** The tag used to identify golf balls in the scene.
- **blackListedTag (static string):** The tag used to mark golf balls as blacklisted, preventing them from being processed further.

**Methods:**

- **GetGolfBalls():**
  - **Returns:** An array of GameObjects tagged as golf balls found in the scene.
- **BlackList(GameObject golfBall):**
  - Marks the specified golf ball as blacklisted by changing its tag to blackListedTag.
- **IsBlackListed(GameObject golfBall):**
  - **Returns:** A boolean indicating whether the specified golf ball is blacklisted.
- **WhiteList(GameObject golfBall):**
  - Marks the specified golf ball as whitelisted by changing its tag back to golfBallTag.

**Important Implementation Details:**

- This class provides a straightforward mechanism for identifying and managing golf balls based on their tags, facilitating the ability to blacklist and whitelist them during gameplay.
- Using tags allows for efficient grouping and retrieval of game objects without the need for additional data structures.

**Use Cases:** The OmnipotentGolfBallSensory class is useful for any gameplay scenario involving golf balls, enabling the dynamic management of their states. This can be applied in scenarios such as collecting or filtering golf balls based on their states in a game.

## 2.12 RandomGolfBallInitializer

The RandomGolfBallInitializer class extends the AbstractGolfBallInitializer class and provides functionality for initializing golf balls with random attributes. It assigns a random level to each golf ball during initialization.

**Methods:**

- **Initialize(GameObject golfBall):**
  - Initializes the specified golf ball by retrieving its GolfBall component and setting its level to a random value from the GolfBallLevel enumeration.

**Important Implementation Details:**

- This class leverages a utility method, `Utils.GetRandomEnumValue<T>()`, to randomly select a value from the GolfBallLevel enum.
- Ensuring that golf balls are initialized with varying levels adds diversity to the gameplay experience.

**Use Cases:** The RandomGolfBallInitializer class is useful in scenarios where golf balls need to be initialized with varying levels, enhancing gameplay mechanics and unpredictability in challenges or obstacles involving golf balls.

## 2.13 Score

The Score class manages the NPC's score in the game. It provides methods to increase and decrease the score, along with an event system to notify other classes of score changes.

**Events:**

- **OnScoreChanged:** A delegate event that other classes can subscribe to in order to be notified when the score changes. It passes the new score value as an integer.

**Fields:**

- **score (int):** Stores the current score of the player, initialized to zero.

**Methods:**

- **IncreaseScore(int amount):**
  - Increases the current score by the specified amount and invokes the OnScoreChanged event with the new score.
- **DecreaseScore(int amount):**
  - Decreases the current score by the specified amount and invokes the OnScoreChanged event with the new score.
- **GetScore():**
  - Returns the current score of the player.

**Important Implementation Details:**

- The score can be both increased and decreased, allowing for versatile score management during gameplay.
- The OnScoreChanged event ensures that any subscribed classes are notified whenever the score changes, allowing for dynamic UI updates or gameplay responses.

**Use Cases:** The Score class is essential for tracking player performance and progress throughout the game, providing functionality to display scores, manage scoring events, and respond to gameplay changes.

## 2.14 SimpleGolfCollectorDecisionMaker

The SimpleGolfCollectorDecisionMaker class determines the optimal golf ball for a GolfBallCollector to collect based on distance. It evaluates all available golf balls and selects the closest reachable one.

**Fields:**

- **collector (GolfBallCollector):** A reference to the GolfBallCollector that this decision maker is associated with, allowing it to evaluate golf balls based on the collector's capabilities.

**Methods:**

- **DecideFor(GolfBallCollector collector):**
  - Sets the associated GolfBallCollector, allowing the decision maker to utilize its properties and methods for decision-making.
- **GetBestGolfBall(GameObject[] golfBalls):**
  - Iterates through the provided array of golf balls, checking their reachability and distance from the collector. It returns the closest reachable golf ball, or null if none are reachable.

**Important Implementation Details:**

- The decision-making process evaluates each golf ball's reachability and distance using the collector's `GetPathDistance` method.
- If a golf ball is unreachable, it is skipped in the evaluation.

**Use Cases:** The SimpleGolfCollectorDecisionMaker class is useful for implementing decision-making logic for golf ball collectors in the game, enabling efficient selection of targets based on proximity and reachability.

## 2.15 SmartGolfCollectorDecisionMaker

The SmartGolfCollectorDecisionMaker class enhances decision-making for golf ball collection by prioritizing golf balls based on their level and distance to the collector and golf cart. It selects the best golf ball based on a combination of difficulty and accessibility.

**Fields:**

- **golfCart (Transform):** A reference to the transform of the golf cart, which is used to calculate distances to the golf balls.
- **n (int):** The number of closest golf balls to consider from each difficulty group. This value must be at least 1.
- **collector (GolfBallCollector):** A reference to the GolfBallCollector that this decision maker is associated with, allowing it to evaluate golf balls based on the collector's capabilities.

**Methods:**

- **DecideFor(GolfBallCollector collector):**
  - Sets the associated GolfBallCollector, allowing the decision maker to utilize its properties and methods for decision-making.
- **GetBestGolfBall(GameObject[] golfBalls):**
  - Groups golf balls by their difficulty level and determines the closest balls in each group. It calculates the accessibility based on the distance from the collector to the golf balls and from the golf balls to the golf cart, ultimately returning the most favorable golf ball for collection.

**Important Implementation Details:**

- The implementation groups golf balls by their level and selects the 'n' closest balls from each group.
- A sorting mechanism is applied to prioritize reachable golf balls, considering both the level of difficulty and the distance.
- The selection process uses a placeholder note about implementing a QuickSelect algorithm for improved performance in future iterations.

**Use Cases:** The SmartGolfCollectorDecisionMaker class is useful for creating more strategic golf ball collection behavior, allowing collectors to optimize their target selection based on difficulty levels and distances, enhancing gameplay dynamics.

## 2.16 SpatialGolfBallInitializer

The SpatialGolfBallInitializer class initializes golf balls based on their proximity to a GolfBallCollector, categorizing them into difficulty levels based on distance thresholds. It adds an element of randomness to the initialization process through a jitter effect, enhancing the variability of golf ball difficulty.

**Fields:**

- **collector (GolfBallCollector):** A reference to the GolfBallCollector, which is used to assess the reachability of golf balls and determine their difficulty based on distance.
- **difficultyThresholds (Vector3):** A vector containing the distance thresholds for categorizing golf balls into Easy, Medium, and Hard levels. The x component represents the easy threshold, the y component represents the medium threshold, and the z component represents the hard threshold.
- **jitter (float):** A positive float that adds randomness to the distance calculations, allowing for slight variations in difficulty assessments.

**Methods:**

- **Initialize(GameObject golfBall):**
  - Checks if the golf ball is reachable from the collector. If reachable, it retrieves the GolfBall component, applies the jitter effect to the calculated distance, and assigns a difficulty level based on the modified distance relative to the defined thresholds.

**Important Implementation Details:**

- The initialization process includes a random offset (jitter) to create variability in difficulty assessments.
- The method uses distance comparisons against the components of the difficultyThresholds vector to set the appropriate GolfBallLevel.

**OnDrawGizmosSelected:**

- Draws wireframe spheres in the Scene view to visualize the difficulty thresholds. The spheres represent the easy (green), medium (yellow), and hard (red) distance ranges from the collector.

**Use Cases:** The SpatialGolfBallInitializer class is useful for dynamically setting the difficulty of golf balls based on their position relative to a collector, contributing to gameplay variability and challenge.

## 2.17 UIManager

The UIManager class manages the user interface elements in the game, including health and score displays, as well as a pause menu. It provides functionalities for updating UI elements based on game state changes and handling user interactions such as restarting the game and resuming gameplay.

**Fields:**

- **Instance (UIManager):** A static instance of the UIManager, used for implementing the singleton pattern to ensure only one instance exists throughout the game.
- **healthSlider (Slider):** A UI slider that visually represents the player's health.
- **scoreText (TextMeshProUGUI):** A text field displaying the current score of the player.
- **restartButton (Button):** A button that allows the player to restart the game.
- **resumeButton (Button):** A button that allows the player to resume the game from a paused state.
- **pauseMenu (GameObject):** A GameObject representing the pause menu UI, which contains options for the player when the game is paused.

**Methods:**

- **Awake():**
  - Initializes the singleton instance of UIManager. If an instance already exists, the new instance is destroyed.
- **Start():**
  - Sets up listeners for the restart and resume buttons, linking them to the respective game management functions.
- **Update():**
  - Checks for user input to toggle the pause menu when the Escape key is pressed.
- **TogglePause():**
  - Toggles the visibility of the pause menu. Pauses or resumes the game based on the menu state.
- **ResumeGame():**
  - Hides the pause menu and resumes the game.
- **UpdateHealthUI(int currentHealth, int maxHealth):**
  - Updates the health slider's value based on the current and maximum health values provided.

- **UpdateScoreUI(int score):**
  - Updates the score text field to display the current score.

**Important Implementation Details:**

- The UIManager employs the singleton pattern to ensure a single instance is accessible globally.
- The UI is updated dynamically based on game state changes through method calls from the GameManager.

**Use Cases:** The UIManager class is essential for managing game user interface elements, facilitating user interactions, and providing real-time feedback on game status such as health and score.

## 2.18 Utils

The Utils class provides utility functions for general use throughout the game. This class is static and contains methods that can be used without creating an instance of the class.

**Methods:**

- **GetRandomEnumValue<T>()**
  - **Type:** `T` where `T : System.Enum`
  - **Returns:** A random value of the specified enum type `T`.
  - **Description:** This method retrieves a random value from the specified enumeration type. It uses the `Random.Range` method to select an index randomly from the array of enum values and returns the corresponding enum value.

**Important Implementation Details:**

- This method is generic, allowing it to work with any enum type, enhancing reusability across different parts of the game.
- The use of `System.Enum` constraints ensures that only enum types can be passed to the method, preventing potential runtime errors.

**Use Cases:** The Utils class is designed for developers who need quick access to utility functions, particularly for randomly selecting values from enums, which can be helpful in various gameplay scenarios such as item drops, character abilities, or other random events.

## 2.19 VisualGolfBallSensory

The `VisualGolfBallSensory` class extends `AbstractGolfBallSensory` and provides functionality to detect visible golf balls based on their position relative to the collector. It employs raycasting to check for visibility within a specified radius and angle.

**Serialized Fields:**

- `radius` (float): The maximum distance at which golf balls can be detected. Minimum value is 0.5.
- `angle` (float): The field of view angle in degrees. The range is between 0 and 360 degrees.

**Static Fields:**

- `golfBallTag` (string): The tag used to identify golf ball objects in the scene.
- `blackListedTag` (string): The tag used to mark blacklisted golf balls.

**Methods:**

- **GameObject[] GetGolfBalls()**
  - **Returns:** An array of `GameObject` instances that are currently visible and tagged as golf balls.
  - **Description:** Finds all game objects tagged with `golfBallTag` and filters them to only include those that are visible according to the defined radius and angle.

- **void BlackList(GameObject golfBall)**
  - **Parameters:** `golfBall` (GameObject): The golf ball to be blacklisted.
  - **Description:** Changes the tag of the specified golf ball to `blackListedTag`.
- **bool IsBlackListed(GameObject golfBall)**
  - **Parameters:** `golfBall` (GameObject): The golf ball to check.
  - **Returns:** `true` if the golf ball is blacklisted; otherwise, `false`.
  - **Description:** Checks if the specified golf ball has been blacklisted by comparing its tag.
- **void WhiteList(GameObject golfBall)**
  - **Parameters:** `golfBall` (GameObject): The golf ball to be whitelisted.
  - **Description:** Changes the tag of the specified golf ball back to `golfBallTag`.
- **bool IsGolfBallVisible(GameObject golfBall)**
  - **Parameters:** `golfBall` (GameObject): The golf ball to check for visibility.
  - **Returns:** `true` if the golf ball is visible within the defined radius and angle; otherwise, `false`.
  - **Description:** Determines if the specified golf ball is visible by checking the distance, angle, and using a raycast to ensure there are no obstructions between the collector and the golf ball.

**Important Implementation Details:**

- The class relies on Unity's physics system for raycasting to determine visibility, ensuring efficient detection of obstacles.
- The radius and angle parameters can be adjusted in the Unity editor to suit gameplay requirements.

**Use Cases:** The `VisualGolfBallSensory` class is ideal for scenarios where the golf ball collector needs to visually identify and interact with golf balls within a specified range and direction, adding a level of realism and strategy to the gameplay.

# 3 Conclusion

This documentation provides a basic overview of the Golf Collector project. For more detailed information, refer to the source code.