Roulette Pro

Thank you for purchasing **Roulette Pro**. Your support is invaluable to us and directly contributes to creating new, exciting games and improving our existing ones. We're committed to delivering high-quality gaming experiences, and your trust helps make that possible.

We look forward to continuing to bring innovative updates and more games to you in the future. If you have any feedback or need assistance, feel free to reach out.

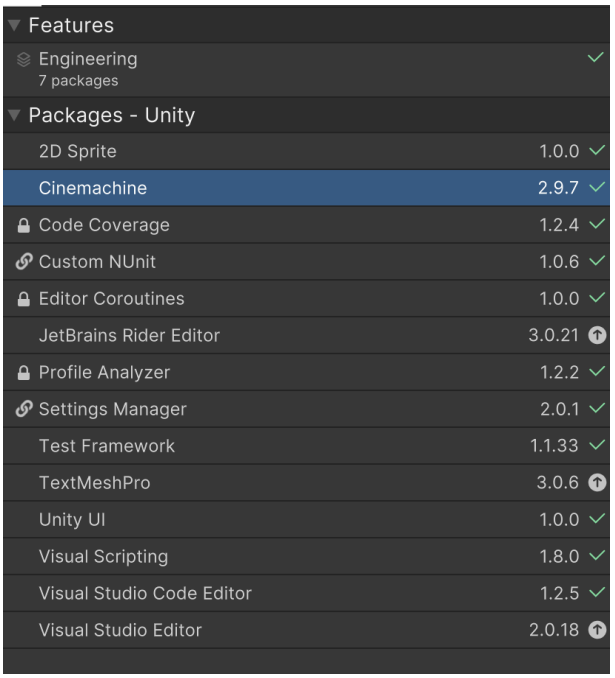
Important - Dependencies

DoTween

To ensure the proper functionality of our **Baccarat** game, it is essential to download the free Unity asset **DOTween** from the Unity Asset Store.

Cinemachine

Use the Official free asset for camera management in Unity. You can go to **Window > Package Manager**, then search ***Cinemachine*** and click Download and Install.



Important - Aspect Ratio

For the desktop version of *Baccarat Pro*, the recommended aspect ratio is **16:9**, ensuring an optimal viewing and gameplay experience. For the mobile version, the recommended resolution is **1080 x 2280**, providing seamless visuals on modern devices.

Introduction

Welcome to the documentation for the Roulette game project, a digital recreation of one of the most popular casino games, built with Unity. This project aims to deliver an engaging and authentic roulette experience for players, leveraging Unity's robust development tools and physics engine to simulate realistic gameplay.

The project features two main variants of roulette: **American Roulette** and **European Roulette**. Each offers distinct characteristics that impact gameplay and strategy.

The game is designed for casino enthusiasts and casual players alike, providing a dynamic and immersive environment, including visually appealing graphics, intuitive controls, and realistic sound effects. This documentation will guide developers, designers, and testers involved in the project through its fundamental rules, structure, and features.

Game Variants and Basic Rules

1. American Roulette

American Roulette features a wheel with **38 slots**, numbered 1 to 36, plus a **single zero (0)** and a **double zero (00)**. The inclusion of the double zero increases the house edge, making this variant more challenging for players.

- **Objective**: Players bet on where they think the ball will land after the wheel is spun.
- **Available Bets**:
    - **Inside Bets**: Specific numbers or small groups of numbers (e.g., straight, split, street).
    - **Outside Bets**: Broader categories (e.g., red/black, odd/even, 1-18/19-36).
- **House Edge**: Approximately **5.26%** due to the double zero.

2. European Roulette

European Roulette has a wheel with **37 slots**, numbered 1 to 36, plus a **single zero (0)**. Without the double zero, this variant offers better odds for players.

1. **Objective**: Same as American Roulette – predicting where the ball will land.
2. **Available Bets**: Identical to American Roulette.
3. **House Edge**: Approximately **2.7%**, making it a preferred choice for many players.

Key Gameplay Features

Betting Options: Players can choose from a variety of betting options, including inside and outside bets.

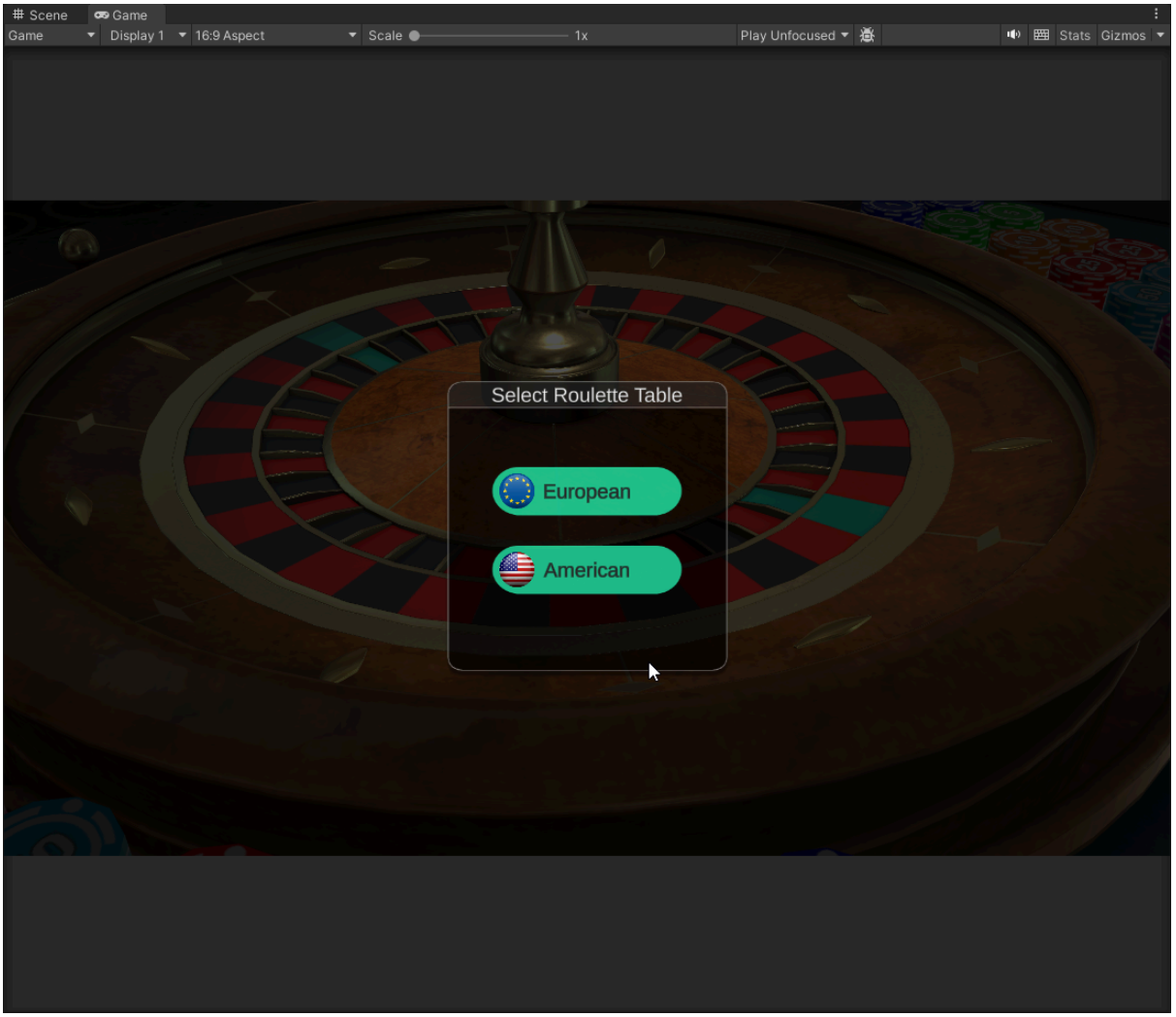
Custom Result System: The outcome of the roulette ball can be determined in three ways: manually by the user, through an API call, or generated randomly by the system. This feature provides flexibility for testing, integration with external systems, or delivering a controlled gameplay experience.

Payout System: The game calculates winnings based on standard roulette payouts (e.g., 35:1 for a straight bet).

Initial Scene: Lobby

Upon launching the game, players are presented with an initial scene where they can choose between the **European** or **American** roulette tables. This selection screen features a clear, intuitive interface with two options, each marked with a corresponding flag for easy identification. Selecting either option will load the respective roulette table, allowing players to experience the unique features and gameplay variations of their chosen version.

This initial scene provides a seamless entry point into the game, ensuring players can quickly start their preferred roulette variant.



The game selection interface features two buttons:

1. **European Roulette**:
    - Redirects players to the **European Mobile Scene**.
    - **Action**: Clicking the button invokes the `GoTo(int sceneIndex)` method with the parameter `1`, which corresponds to the **Euro_mobile** scene.
2. **American Roulette**:
    - Redirects players to the **American Mobile Scene**.
    - **Action**: Clicking the button invokes the `GoTo(int sceneIndex)` method with the parameter `2`, which corresponds to the **American_mobile** scene.

The buttons are clearly labeled with text and icons, ensuring a user-friendly experience.

Scene Management

The `GoToScene` script handles the navigation between scenes using Unity's **SceneManager**. The script provides two methods for loading scenes:

1. **GoTo(string sceneName)**:
    - Loads a scene based on its name.
    - This method allows for more readable code but requires the exact scene name as a parameter.
2. **GoTo(int sceneIndex)**:
    - Loads a scene based on its index in the **Build Settings**.
    - The scene index corresponds to the order of the scenes in Unity's **Scenes In Build** list:
        - **Lobby (Index 0)**: The default entry scene where players choose the game type.
        - **Euro_mobile (Index 1)**: The European roulette mobile scene.
        - **American_mobile (Index 2)**: The American roulette mobile scene.

Important Note: The buttons in the lobby must be configured in the Unity Editor to pass the correct integer value (`1` for European and `2` for American) to the `GoTo(int sceneIndex)` method via Unity Events.

Game Scene: American Roulette

In the American Roulette scene, players are presented with a classic American-style roulette table layout. This setup includes the distinctive **double zero (00)** slot alongside the single zero (0), providing the authentic experience unique to American roulette. The table's cloth displays all possible betting options, including inside and outside bets, allowing players to place chips on individual numbers, colors, or groups. Additional interface elements, such as chip denominations and bet control buttons, give players complete control over their wagers. The scene also includes a display of "Hot" and "Cold" numbers, enhancing gameplay by showing recent outcomes and statistics.



Game Scene: European Roulette

The European Roulette scene features the traditional European-style layout, where the table includes a single zero (0) slot. This setup provides a slightly better chance for players compared to the American version due to the absence of the double zero (00). The table's cloth displays all the standard betting options, including inside and outside bets, allowing players to place chips on individual numbers, colors, and ranges. As with the American scene, this setup includes interactive elements for chip selection, betting controls, and a "Hot" and "Cold" numbers display, which provides statistical insights based on recent results.



Hot and Cold Numbers and History Panels

The **Hot and Cold Numbers Panel** provides valuable statistical information for players, highlighting the numbers that have appeared most frequently (Hot Numbers) and those that have appeared the least (Cold Numbers) during recent spins. This panel not only serves as a visual guide for potential betting strategies but also allows for direct interaction; players can click on a number within this panel to automatically highlight it on the betting table, making it easy to place bets based on recent trends.

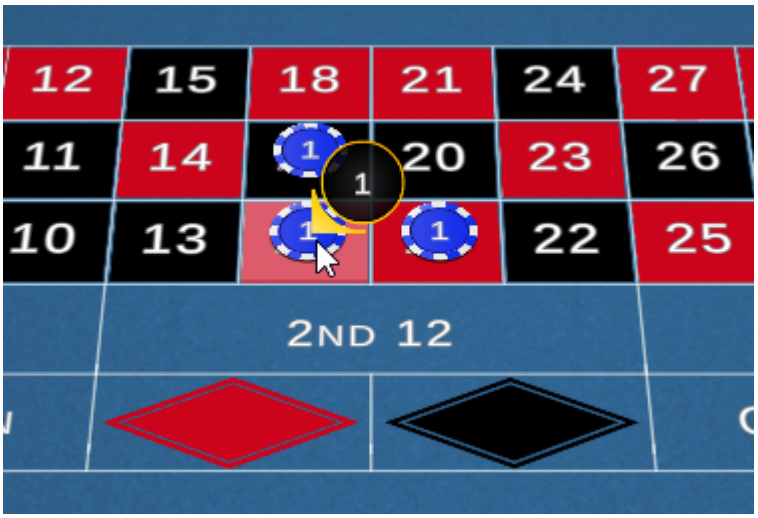
To the right of the Hot and Cold panel, the **History Panel** displays the sequence of previously spun numbers in the order they appeared, with each number shown in its corresponding color (red, black, or green). This panel helps players track the game's flow, offering insights into recent outcomes and assisting with strategies based on patterns in previous spins.



Betting Cloth and Tooltip

The **Betting Cloth** is where players place their chips to make bets. It displays a detailed grid of numbers, colors, and sections that allow for various types of bets, such as individual numbers, splits, streets, corners, and more. Players can easily place chips on specific areas of the cloth to indicate their wagers, with intuitive feedback for positioning.

When a chip is placed on the cloth, a **Tooltip** appears, showing the total value of the current bet on that spot. This tooltip helps players keep track of their individual bet amounts on different areas of the table, providing a clear overview of their stakes before they spin the wheel. In the image above, for example, the tooltip shows a value of "1" for each chip placed on the number 1, making it easy to monitor and adjust bets as needed.

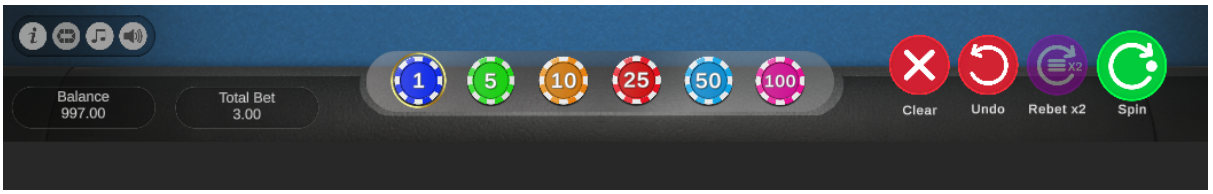


Main Control Panel

The **Main Control Panel** provides players with all essential tools for managing their bets and controlling gameplay. This panel is located at the bottom of the screen and contains the following elements:

- **Chip Selection**: Players can choose from a range of chip denominations (1, 5, 10, 25, 50, and 100) to place on the betting cloth. Selecting a chip value allows players to adjust their bet amount quickly and accurately.
- **Bet Management**:
    - **Clear**: Removes all bets from the table, allowing players to start fresh with their betting layout.
    - **Undo**: Reverses the last bet placement, providing flexibility in adjusting wagers.
    - **Rebet x2**: Repeats the previous round's bet and doubles the amount, making it easy to maintain a betting pattern with increased stakes.
    - **Rebet**: Repeats the previous round's bet. It appears over the Spin Button after each round.
- **Spin Button**: Once the bets are placed, players can press the Spin button to start the game and see the outcome of their bets.
- **Bet Display**:
    - **Balance**: Shows the player's remaining funds, helping them keep track of their available balance.
    - **Total Bet**: Displays the total amount wagered for the current round, providing a clear summary of the player's active bets.

This panel serves as the primary interface for placing and managing bets, giving players convenient access to all key functions needed for a smooth and enjoyable roulette experience.



Neighbors Bet Panel

The **Neighbors Bet Panel** allows players to place bets on a number and its adjacent neighbors on the roulette wheel. This option provides a unique betting style that covers multiple numbers in a sequence, making it ideal for players who want to increase their odds of winning by covering a broader area around a specific number.

In the image, you can see how selecting a number highlights it along with its neighbors directly on the betting cloth. The player can adjust the **offset** value (from 1 up to 8), which determines the number of neighbors included on each side of the main number. For example, selecting an offset of 1 will highlight the chosen number along with one number to its left and right on the wheel, creating a three-number bet. Similarly, an offset of 8 would cover the main number plus eight numbers on both sides, resulting in a larger betting area.

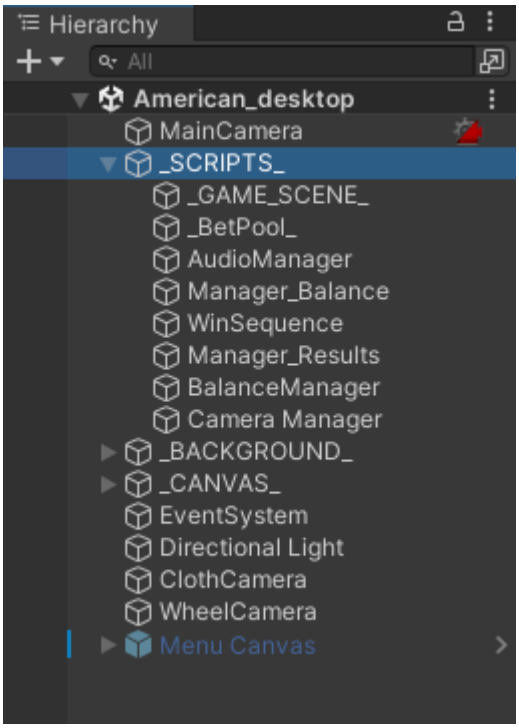
This feature is commonly used in European roulette, where players often bet on groups like **Tiers**, **Orphelins**, **Voisins**, or **Zero** sections, which are based on specific regions of the roulette wheel.



Functionality

SCRIPTS

The ***SCRIPTS*** object contains all essential systems and managers that control the core functionality of the roulette game. These scripts are responsible for various aspects of game management, including bets, balance tracking, result processing, and more. Each script under this object plays a specific role in ensuring smooth gameplay and accurate game logic.



SceneRoulette

The **SceneRoulette** class serves as the main controller for managing the game's user interface, betting interactions, and game state in the roulette scene. It integrates various UI elements, manages player interactions, and communicates with other game systems like the betting pool and the roulette wheel. Below is a breakdown of its functionality:

- **Singleton Instance**: `SceneRoulette` uses a static `_Instance` variable, allowing it to act as a singleton, making it easily accessible throughout the game.
- **UI Elements**:
    - **Balance and Bet Display**: Displays the player's current balance (`textBalance`) and total bet (`textBet`). These are updated dynamically to reflect changes in bets and results.

    - **Result Text**: Shows the outcome of each spin.
    - **Glow Effect**: A visual effect that highlights winnings to the player.
- **Buttons**:
    - **Clear Button**: Clears all current bets when pressed.
    - **Undo Button**: Removes the last bet placed.
    - **Rebet Button**: Repeats the previous bets.
    - **Double Rebet Button**: Repeats the previous bets with doubled values.
    - **Roll Button**: Initiates the roulette spin.
- **Game State Management**:
    - `uiState` controls the visibility of popup windows.
    - `GameStarted` and `MenuOn` flags manage whether bets can be placed and whether the game is active.
    - `WaitTime` sets a delay for certain actions.
- **Core Methods**:
    - **OnButtonClear**, **OnButtonUndo**, **OnButtonRebet**, **OnButtonDoubleRebet**: Handle respective button interactions, triggering sounds and updating the betting pool.
    - **OnButtonRoll**: Initiates a roulette spin and disables buttons to prevent further bets until the spin completes.
    - **SpinRoulette**: Starts the wheel spin, plays a sound, and updates the UI to reflect the game state.
    - **ChangeUI**: Disables betting-related UI elements during a spin.
    - **BlockBets** and **ReleaseBets**: Toggle the ability to place bets based on the game state.
    - **UpdateLocalPlayerText**: Updates the displayed balance and bet amount for the player.
    - **UpdateWin**: Displays winnings and triggers a glow effect animation to highlight the win.

BetPool

The **BetPool** class is responsible for managing all bet placements in the roulette game, including tracking, clearing, undoing, and rebetting. It acts as a central system for handling player bets, storing information about each bet, and providing functionality to reset, modify, or repeat bets.

Key Components

- **Singleton Instance**: `BetPool` uses a singleton pattern via the `Instance` variable, making it globally accessible and ensuring only one instance exists.
- **Data Structures**:
    - **_BetFootprints**: A list that stores all individual bet entries as `BetFootprint` objects, which contain information about each bet, including the bet space, amount, and transaction ID.
    - **_BetsList**: A list of active `BetSpace` objects that currently have bets placed on them.
    - **_RebetList**: A copy of `_BetsList` used to restore the previous round's bets when rebetting.

Core Methods

- **Awake**: Initializes the lists `_BetFootprints` and `_BetsList`. If an instance already exists, it destroys any duplicate `BetPool` object.
- **ResetStatus**: Clears `_BetFootprints`, copies `_BetsList` to `_RebetList` for rebetting, and clears `_BetsList`.
- **Add**: Adds a new bet to `_BetFootprints` and updates `_BetsList` if the `BetSpace` is not already included. This method records each bet along with its value and transaction ID, enabling tracking and future modifications.
- **Clear**: Clears all current bets by iterating over `_BetsList` and calling each `BetSpace`'s `Clear` method. It then clears `_BetFootprints` and `_BetsList` and disables the **Roll**, **Undo**, and **Clear** buttons on the `SceneRoulette` UI. The total bet is reset to zero.
- **Undo**: Removes the last bet placed by accessing the most recent `BetFootprint` in `_BetFootprints`. It finds and removes all bets with the same transaction ID, iterating through each footprint to remove the corresponding value from the `BetSpace`. If the bet amount on a space reaches zero, it is removed from `_BetsList`. This method also updates the UI buttons based on remaining bets.
- **Rebet**: Repeats the last round's bets by iterating over `_RebetList` and calling the `Rebet` method on each `BetSpace`. It takes an optional `multi` parameter to

multiply the bet amount, allowing players to rebet with increased stakes. The UI buttons are updated after rebets are placed.

BetFootprint Class

The **BetFootprint** class represents individual bet entries and includes:

- **Id**: A unique identifier (GUID) for each transaction, allowing grouped bet operations.
- **betSpace**: The specific `BetSpace` object where the bet is placed.
- **value**: The bet amount.

BalanceManager

The **BalanceManager** class manages the player's balance in the roulette game. It provides functionality to set, update, and reset the balance, ensuring that changes are immediately reflected in the user interface.

Key Components

- **Balance Property**:
  - `Balance` is a static property that holds the player's current balance. It is private for setting to ensure only authorized methods modify it, while allowing public read access.

Core Methods

- **SetBalance(float balance)**:
  - Sets the player's balance to a specified amount. This method is typically used to initialize the balance at the start of the game or after a reset. After setting the balance, it calls `SceneRoulette.UpdateLocalPlayerText()` to update the balance display in the game UI.
- **ChangeBalance(float value)**:
  - Adjusts the balance by a specified amount, either adding or subtracting from the current balance (e.g., deducting a bet or adding winnings). The balance change is immediately reflected in the UI by calling `SceneRoulette.UpdateLocalPlayerText()`.
- **ResetBalance(float balance)**:

    - Resets the balance to a specified amount. This instance method can be used to reset the player's balance mid-game if needed, again updating the UI to show the new balance.

BallManager

The **BallManager** class controls the behavior and animation of the roulette ball during gameplay. It manages the ball's spinning, movement, and landing on the final result, simulating a realistic experience for players. This class is crucial for implementing custom or random results, as it enables precise control over where the ball lands.

Key Components

- **Ball and Wheel References**:
    - `ball`: A `Rigidbody` representing the physical ball in the game.
    - `wheel`: A reference to the `Wheel` class, which contains the roulette wheel information, including potential result positions.
    - `resultPoint`, `originPoint`: Transforms that define the ball's starting and landing points.
    - `pivotTransform`, `pivotWheelTransform`: Transforms used for positioning the ball relative to the wheel during spins.
- **Spin Control Variables**:
    - `angularSpeed`: Controls the speed of the ball's rotation around the wheel.
    - `spinning`, `stopping`, `bouncing`: Boolean flags that manage the spinning and stopping process, as well as bouncing effects when the ball lands.
    - `trigger_animateBall`: A flag to trigger the animation when the ball reaches its target.
- **Camera and Animation**:
    - `camCtrl`: A reference to `CameraManager` for switching camera views.
    - `popUpAnimations`: A list of animations shown or hidden during gameplay.

Core Methods

- **StartSpin**:
    - Initializes the ball for a new spin by setting it to the origin point, resetting its rotation and speed, and enabling the `spinning` flag. It hides any pop-up animations at the start of the spin.

- **FindNumber**:
    - This method is used to select the final result for the ball. By taking an integer `result`, it sets the target location (`Target`) for the ball on the wheel. It gradually reduces the ball's speed using DOTween, eventually reaching a stop, where the ball lands on the desired number.
    - **Significance**: This function enables the ability to control the outcome of the spin. By defining the `result`, you can use an API or a random number generator to specify where the ball will land, making it possible to predetermine results.
- **PlaceToResult**:
    - Positions the ball at the result point with a jumping animation that mimics bouncing, enhancing realism when the ball lands on its final spot. This method starts a coroutine to play a bouncing sound during the animation.
- **BounceSound Coroutine**:
    - Plays a sound effect repeatedly while the ball is bouncing, adding an auditory cue to complement the visual landing effect.
- **CalculateAngleRatio**:
    - Computes the angle between the ball's current direction and the target position, which helps to gradually align the ball with its intended result on the wheel.
- **FixedUpdate**:
    - Called at fixed intervals, this method rotates the ball around the wheel while `spinning` is true. When `stopping` is enabled, it uses angle calculations to slow down the ball and guide it to the target position. Once the ball is aligned with its target (`Target`), `PlaceToResult` is called to finalize the animation, and spinning is disabled.
    - After stopping, the ball is released from its rotation, camera view switches to focus on the cloth, and the result is recorded by `resultManager`.

Custom Result Control

With the **FindNumber** method, this class allows the ball's final resting position to be set programmatically. This feature is essential for:

- **API Integration**: The result can be fetched from an external API, enabling external systems to define the outcome.
- **Randomization**: Any random number generation function can provide a valid roulette number, which `FindNumber` will use to determine the ball's target position.

The **BallManager** class, therefore, provides both flexibility and control, allowing for a mix of predetermined and randomized outcomes, enhancing the versatility and realism of the roulette gameplay experience.

Mobile menú components



1. **Info Button (Leftmost Icon)**:
   - **Purpose**: Opens a help or tutorial section where players can find information about the game rules, betting mechanics, and general gameplay guidance.
   - **Icon**: Represented by an “i” symbol, easily recognizable as an information section.
2. **Audio Button (Speaker Icon)**:
   - **Purpose**: Toggles the game’s audio on or off. This allows players to quickly manage sound effects and background music without navigating to a settings menu.
   - **Icon**: A speaker symbol indicating the current audio state (muted or unmuted).
3. **Balance Display**:
   - **Purpose**: Shows the player’s current balance, providing real-time updates as bets are placed or winnings are added.
   - **Location**: Center-left of the menu for easy visibility during gameplay.
4. **Total Bet Display**:
   - **Purpose**: Displays the total amount wagered in the current round, allowing players to track their bets easily.
   - **Location**: Center-right of the menu, paired with the balance for clarity.
5. **Neighbors Menu (Chip Icon)**:
   - **Purpose**: Opens the **Neighbors Menu**, where players can place bets on numbers adjacent to a selected number on the roulette wheel. This feature allows for strategic betting by covering groups of numbers around a central target.

- **Icon**: A chip symbol that represents betting functionality, emphasizing its connection to the Neighbors menu.

6. **Hot and Cold Menu (Bars Icon)**:

- **Purpose**: Opens a submenu displaying **Hot and Cold Numbers** and the game's current status. This feature provides insights into the most and least frequently occurring numbers and allows players to strategize their bets based on statistical trends.

- **Icon**: Resembles signal strength bars, but it represents the game's **Hot and Cold Numbers** feature and the state of the game.

AudioManager

The **AudioManager** class is responsible for managing and playing audio in the game, including background music (BGM) and sound effects (SFX). It uses a pooling system for efficient management of audio sources, ensuring smooth playback and minimal performance overhead.



Key Features

1. **Singleton Pattern**:
   - The class uses a static `_Instance` to ensure there is only one instance of the **AudioManager**, accessible globally.

2. **Audio Source Pool**:
    - The `AudioSourcePool` maintains a list of reusable `AudioSource` objects to prevent creating and destroying audio components repeatedly, improving performance.
    - `AudioSourceAlloc()`: Initializes the pool with `maxAudioSourcePool` (default: 15) `AudioSource` objects.
3. **Sound Playback**:
    - **SoundPlay(int iType)**: Plays a sound effect based on the specified index (`iType`) in the `audioEvent` array.
    - **SoundPlayIn(int iType, float fDelay)**: Plays a sound effect after a delay.
    - **StopAuxiliar()**: Stops the auxiliary sound effect, typically used for long-playing or looping effects (e.g., rolling sounds).
4. **Volume Control**:
    - **ChangeVolume()**: Adjusts the volume of the background music based on `MusicVolume`.
    - **GetVolume(string key)**: Retrieves the current volume level for a specified audio mixer parameter.
    - **AdjustVolume(string key, float volume)**: Adjusts the volume of a specific audio mixer parameter.
    - **ToggleVolume(string key, bool isOn)**: Toggles audio on or off by setting the parameter to either a normal level (0 dB) or muted (-80 dB).
5. **Auxiliary Audio**:
    - `auxiliarAS`: A specific `AudioSource` for handling looping or auxiliary sounds, such as background effects.

Playing a Sound Effect:
AudioManager.SoundPlay(3); // Play Sound

Changing Volume:
AudioManager.AdjustVolume("Master/SFX", 0.5f); // Sets the SFX volume to 50%.

AudioEvent

The **AudioEvent** class is a **ScriptableObject** that defines the properties of individual sounds in the game. It simplifies the creation and configuration of sound effects by allowing audio properties to be stored and reused across multiple scenes or components.

Key Features

1. **Properties**:
    - **AudioClip**: The sound file to be played.
    - **AudioMixerGroup**: The mixer group for routing audio (e.g., SFX, BGM).
    - **Volume and Pitch**: Adjustable parameters for fine-tuning the sound.
    - **Looping**: Boolean to determine if the sound should loop.
2. **Playback Methods**:
    - **PlayIn(AudioSource source)**: Plays the clip with default properties.
    - **PlayIn(AudioSource source, float volume)**: Plays the clip with a custom volume.

Steps to Create and Use an AudioEvent

1. **Creating an AudioEvent**:
    - Right-click in the **Assets** panel in Unity.
    - Navigate to **Create > Sound > SoundEvent**.
    - Name the new asset and configure its properties:
        - Drag and drop the desired audio clip into the `clip` field.
        - Assign the appropriate `AudioMixerGroup` (e.g., SFX or BGM).
        - Set the desired `volume`, `pitch`, and whether the sound should loop.
2. **Using AudioEvent in AudioManager**:
    - Add the created `AudioEvent` assets to the `audioEvent` array in the **AudioManager** script.
    - Reference the index of the desired sound in the array when calling `SoundPlay`.

CameraManager

The **CameraManager** class is responsible for switching between different virtual cameras in the game, ensuring dynamic and smooth transitions based on the game state. This implementation leverages **Cinemachine Virtual Cameras**, a powerful Unity tool for managing camera behavior.

Key Features

1. **Cinemachine Integration**:
    - Uses **CinemachineVirtualCamera** components to define different camera views:
        - `_clothCamera`: Focuses on the roulette betting cloth, used during the betting phase.

        - `_rouletteCamera`: Focuses on the roulette wheel, used during the spin phase.
2. **Priority-Based Camera Switching**:
    - Adjusts the **Priority** of each camera to determine which one is active. Higher priority cameras take precedence over lower priority ones.

Core Methods

1. **SwitchToClothCamera()**:
    - Sets the `_clothCamera` priority to **10**, making it the active camera.
    - Sets the `_rouletteCamera` priority to **0**, deactivating it.
    - **Use Case**: Called during the betting phase when the focus should be on the betting cloth.

2. **SwitchToRouletteCamera()**:
    - Sets the `_rouletteCamera` priority to **10**, making it the active camera.
    - Sets the `_clothCamera` priority to **0**, deactivating it.
    - **Use Case**: Called during the spinning phase when the focus should be on the roulette wheel.

Implementation Steps

1. **Cinemachine Setup**:
    - Add **CinemachineVirtualCamera** components to the scene.
    - Assign `_clothCamera` to the virtual camera focusing on the betting cloth.
    - Assign `_rouletteCamera` to the virtual camera focusing on the roulette wheel.
2. **Assign Cameras**:
    - Drag and drop the respective CinemachineVirtualCamera objects into the `_clothCamera` and `_rouletteCamera` fields in the **CameraManager** component in the Unity Inspector.
3. **Call Methods Based on Game State**:
    - Call `SwitchToClothCamera()` during the betting phase.
    - Call `SwitchToRouletteCamera()` during the spinning phase.