Waypoint Indicator Basic: User Guide

This package is a user-friendly and customizable target indicator system designed to easily and flexibly display in-game objectives. The system works with the main components `IndicatorRenderer` and `IndicatorController`. `IndicatorRenderer` can be used independently, while `IndicatorController` is developed to provide easier control and management.

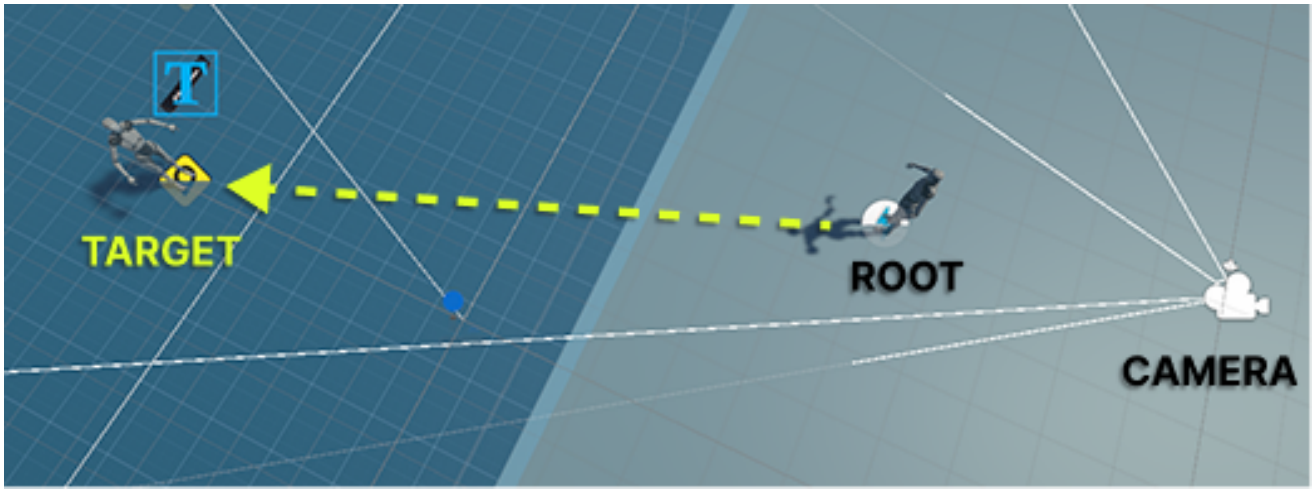
Quick Start

To start using the Waypoint Indicator Basic system, follow these steps:

Step 1: Adding `IndicatorManager` to the Scene

The `IndicatorManager` is the central hub for all indicators and there should only be one instance of it in the scene. You can do this in two ways:

1. **Using a Prefab:** Drag and drop the "Canvas" prefab from the `Prefabs` folder into your scene. This prefab contains all the necessary components and settings.
2. **Manual Setup:**
    - Create a new empty GameObject and rename it to "IndicatorManager".
    - Add the `IndicatorManager` script to this GameObject.
    - Assign an existing or newly created UI Canvas in your scene to the `Canvas Parent` field of the `IndicatorManager` component. This will be the main UI element where indicators are created.
    - Assign an `IndicatorProfile` asset (which you can create from the `Scriptable Objects/IndicatorProfile` menu) to the `Indicator Profile` field. This profile defines the prefabs to be used for different indicator types.
    - Assign the `Transform` component of your player character or main camera to the `Root Transform` field for accurate distance calculations.

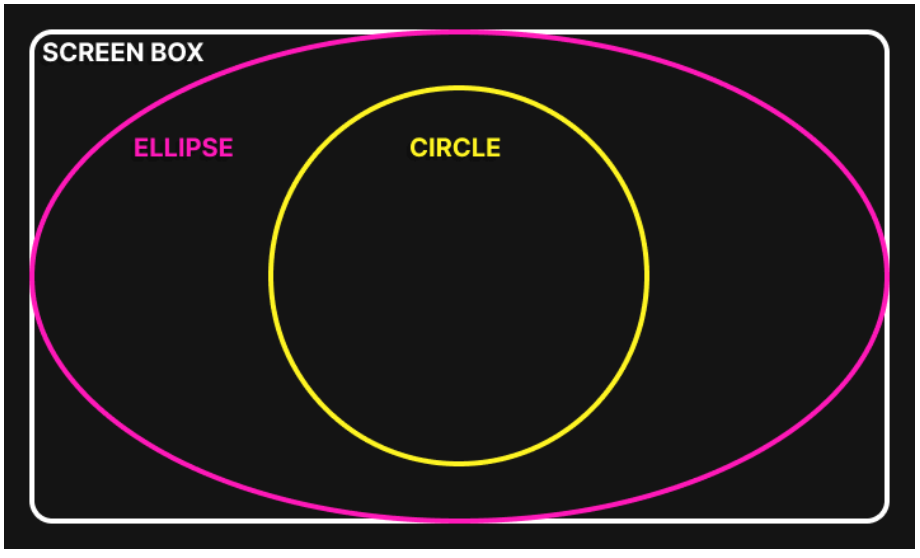


Step 2: Adding the `IndicatorController` Component to the Target

Add the `IndicatorController` script to any target GameObject you wish to display an indicator for. This component provides an easy way to control the `IndicatorRenderer`.

Important `IndicatorController` variables and their descriptions:

- **Indicator Type:** Determines the type of indicator prefab to use (e.g., Waypoint, Enemy, Objective, Custom).
- **Custom Indicator Name:** When `Indicator Type` is set to "Custom", enter the name of your custom indicator defined in the `IndicatorProfile` here.
- **Offset:** Adjusts the position offset of the indicator relative to the target.
- **Image:** The main image of the indicator.
- **Image Color:** The color of the indicator image.
- **Compass Image:** The image to be displayed on the compass.
- **Compass Image Color:** The color of the compass image.
- **Level:** The level or number to be displayed on the indicator.
- **Slider Color:** The color of the slider (e.g., health bar).
- **Max Health:** The maximum value of the slider (commonly used for health bars).
- **Indicator Text:** The text to be displayed on the indicator.
- **Override Settings:** Determines whether this indicator will override the global settings in `IndicatorRenderer` and use its own settings.
- **Show On Compass:** Determines whether the indicator will appear on the compass.
- **Show Off Screen:** Determines whether the indicator will appear when the target is off-screen.
- **Show Direction Arrow:** Determines whether a direction arrow will be shown when the target is off-screen.
- **Hide By Distance:** Determines whether the indicator will be hidden based on distance.
- **Show Distance:** Determines whether the distance to the target will be shown.
- **Show Off Screen Distance:** Determines whether the distance will be shown when the target is off-screen.
- **Show Decimal:** Determines whether the distance will be displayed with decimal places.
- **Min Distance:** The minimum distance at which the indicator starts to appear.
- **Max Distance:** The maximum distance at which the indicator remains visible.
- **Screen Padding:** The padding of the indicator from the screen edges.
- **Distance Format:** Determines the distance unit (e.g., metric, imperial).
- **Edge Mode:** Determines how the indicator behaves at the screen edges.



Now, your targets will automatically be displayed thanks to the `IndicatorController` component added to them!

Step 3: Interacting with `IndicatorController` via Scripts

`IndicatorController` provides various functions to dynamically control the indicator from your scripts.

`TestButton.cs` **Example:** This example script changes the indicator type to "Interaction" when the player is near the target and fills a slider when the 'E' key is pressed. When the player moves away, it reverts the type back to "Objective".

C#

```
// TestButton.cs
// ...
void Update()
{
    if (Vector3.Distance(transform.position, player.transform.position) < 3)
    {
        if (controller.indicatorType != IndicatorType.Interaction)
        {
            controller.ChangeType(IndicatorType.Interaction);
        }
        else if (Input.GetKeyDown(KeyCode.E))
        {
            print("Button pressed");
        }
        value = Mathf.Clamp(value + (Input.GetKey(KeyCode.E) ? Time.deltaTime : -Time.deltaTime), 0f, 1f);
        controller.SetSliderValue(value);
    }
    else
    {
        if (controller.indicatorType != IndicatorType.Objective)
        {
            controller.ChangeType(IndicatorType.Objective);
        }
    }
}
// ...
```

`TestEnemy.cs` **Example:** This script demonstrates how to update an enemy's health bar using `IndicatorController`. The 'E' key decreases the enemy's health, and if health reaches zero, the enemy is destroyed.

C#

```
// TestEnemy.cs
// ...
void Start()
{
    controller.SetSliderValue(0, maxHealth, health);
}

void Update()
{
    if (Vector3.Distance(player.position, transform.position) < 3f)
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            health = Mathf.Clamp(health - 10, 0, maxHealth);
            if (health == 0)
                Destroy(gameObject);
            else
                controller.SetSliderValue(health);
        }
    }
}
// ...
```

Step 4: Compass Setup

If you performed a manual setup and wish to use the compass:

- Add the "Compass" prefab from the `Prefabs` folder under your UI Canvas in the scene.
- By checking the `Show On Compass` option on the `IndicatorController`, you can make the respective targets appear on the compass.

Detailed User Manual

`IndicatorController` **Variables and Functions**

The following tables describe all the variables and functions of the `IndicatorController` class.

Variables

Variable Name	Description
`indicatorType`	Determines the type of indicator (e.g., Waypoint, Enemy). This type is used to select prefabs from the `IndicatorProfile`.
`customIndicatorName`	Specifies the name of the custom indicator to be used when `indicatorType` is "Custom".
`offset`	Adjusts the position offset of the indicator relative to the target GameObject.
`image`	The main `Sprite` image used in the indicator UI.
`compassImage`	The `Sprite` image to be displayed on the compass.
`level`	The level or number displayed on the indicator.
`imageColor`	The color of the main image `Sprite`.

compassImageColor	The color of the compass image Sprite.
sliderColor	The color of the slider (e.g., health bar).
maxHealt	The maximum value of the slider. Typically used for health bars.
indicatorText	The text displayed on the indicator.
overrideSettings	Determines whether this indicator will override the global IndicatorManager settings and use its own.
showOnCompass	Determines whether the indicator will appear on the compass.
showOffScreen	Determines whether the indicator will appear when the target is off-screen.
showDirectionArrow	Determines whether a direction arrow will be shown when the target is off-screen.
hideByDistance	Determines whether the indicator will be hidden based on distance.
showDistance	Determines whether the distance to the target will be shown.
showOffScreenDistance	Determines whether the distance will be shown when the target is off-screen.
showDecimal	Determines whether the distance will be displayed with decimal places.

minDistance	The minimum distance at which the indicator starts to appear.
maxDistance	The maximum distance at which the indicator remains visible.
screenPadding	The padding of the indicator from the screen edges.
distanceFormat	Determines the distance unit (e.g., metric, imperial).
edgeMode	Determines how the indicator behaves at the screen edges (e.g., Clamp, Hide).

Functions

Function Name	Description	Usage Example
SetOffset(Vector3 offset)	Sets the offset of the indicator relative to the target GameObject.	controller.SetOffset(new Vector3(0, 2, 0));
ChangeType(IndicatorType type)	Changes the indicator type and re-instantiates the prefab according to the new type.	controller.ChangeType(IndicatorType.Enemy);
SetImage(Sprite image)	Sets the main image of the indicator.	controller.SetImage(myCustomSprite);
SetCompassImage(Sprite image)	Sets the image to be displayed on the compass.	controller.SetCompassImage(myCompassSprite);

SetLevel(int level)	Sets the level/number value displayed on the indicator.	controller.SetLevel(5);
SetSliderValue(float currentValue)	Sets the current value of the slider (maxHealth value is preserved).	controller.SetSliderValue(75);
SetSliderValue(float maxValue, float currentValue)	Sets both the maximum and current values of the slider.	controller.SetSliderValue(100, 75);
SetText(string txt)	Sets the text displayed on the indicator.	controller.SetText("Quest Objective");
SetImageColor(Color color)	Sets the color of the indicator's main image.	controller.SetImageColor(Color.blue);
SetSliderColor(Color color)	Sets the color of the slider.	controller.SetSliderColor(Color.green);
SetVisibility(bool value)	Enables or disables the visibility of the indicator.	controller.SetVisibility(true);
SetDistanceVisibility(bool value)	Enables or disables the visibility of the distance display.	controller.SetDistanceVisibility(false);
SetDistanceRange(float min, float max)	Sets the minimum and maximum distance range for the indicator to be visible.	controller.SetDistanceRange(5f, 50f);

SetScreenPadding(Vector2 padding)	Sets the padding amount from the screen edges.	controller.SetScreenPadding(new Vector2(100, 100));
SetDistanceFormat(IndicatorRenderer.unit format)	Sets the distance unit format.	controller.SetDistanceFormat(IndicatorRenderer.unit.imperial);
SetEdgeMode(IndicatorRenderer.IndicatorEdgeMode mode)	Sets the behavior mode of the indicator at screen edges.	controller.SetEdgeMode(IndicatorRenderer.IndicatorEdgeMode.Clamp);
SetShowDirectionArrow(bool value)	Enables or disables the display of the direction arrow.	controller.SetShowDirectionArrow(true);
SetShowOffScreen(bool value)	Enables or disables the indicator's visibility when off-screen.	controller.SetShowOffScreen(true);
SetHideByDistance(bool value)	Enables or disables the hide-by-distance feature.	controller.SetHideByDistance(true);
SetShowOffScreenDistance(bool value)	Enables or disables the display of distance when off-screen.	controller.SetShowOffScreenDistance(true);
SetShowDecimal(bool value)	Enables or disables the display of decimal places in the distance.	controller.SetShowDecimal(false);
SetShowOnCompass(bool value)	Enables or disables the indicator's visibility on the compass.	controller.SetShowOnCompass(true);

`IndicatorRenderer` Variables and Functions

The `IndicatorRenderer` is the indicator UI itself and is responsible for basic visibility, positioning, and visual settings.

Variables

Variable Name	Description
`root`	The root transform to be used when the indicator follows the target (usually camera or player).
`target`	The transform of the target GameObject that the indicator will follow.
`offset`	The position offset of the indicator relative to the target.
`indicatorName`	The name given to the indicator (for internal use or debugging).
`source`	The `CanvasGroup` component of the indicator UI, used for fade effects.
`directionArrow`	The UI element for the direction arrow when the target is off-screen.
`image`	The main `Sprite` image of the indicator.
`imageColor`	The color of the main image `Sprite`.
`showOnCompass`	Whether the indicator will appear on the compass.
`compassImage`	The `Sprite` image to be displayed on the compass.

compassImageColor	The color of the compass image Sprite.
fadeTime	The duration of the fade effect for indicator visibility changes (enter/exit, visible/invisible).
showDirectionArrow	Whether the direction arrow will be shown when off-screen.
showOffScreen	Whether the indicator will appear when the target is off-screen.
hideByDistance	Whether the indicator will be hidden based on distance.
minDistance	The minimum distance at which the indicator starts to appear.
maxDistance	The maximum distance at which the indicator remains visible.
edgeMode	How the indicator behaves at the screen edges (e.g., Clamp, Hide).
screenPadding	The padding of the indicator from the screen edges.
distanceText	The TextMeshPro component displaying the distance.
showDistance	Whether the distance to the target will be shown.
showOffScreenDistance	Whether the distance will be shown when the target is off-screen.
showDecimal	Whether decimal places will be shown in the distance.

distanceFormat	The distance unit format (e.g., metric, imperial).
indicatorText	The TextMeshPro component displaying the main text on the indicator.
text	The text displayed on the indicator.
levelText	The TextMeshPro component displaying the level text.
level	The level or number displayed on the indicator.
slider	The Slider UI component.
maxValue	The maximum value of the slider.
currentValue	The current value of the slider.
enableDebug	Enables/disables debug messages.

Functions

Function Name	Description	Usage Example
SetText(string value)	Sets the text displayed on the indicator.	indicator.SetText("Target Nearby");
SetLevel(int value)	Sets the level/number value displayed on the indicator.	indicator.SetLevel(10);

`SetImageColor(Color color)`	Sets the color of the indicator's main image.	`indicator.SetImageColor(Color.red);`
`SetSliderValue(float current, float max)`	Sets the current and maximum values of the slider.	`indicator.SetSliderValue(50, 100);`
`SetVisible(bool value)`	Sets the permanent visibility of the indicator.	`indicator.SetVisible(false);`
`OnEnterScreen()`	Virtual function called when the indicator enters the screen.	(To be overridden in derived classes)
`OnExitScreen()`	Virtual function called when the indicator exits the screen.	(To be overridden in derived classes)
`OnVisible()`	Virtual function called when the indicator becomes visible.	(To be overridden in derived classes)
`OnInvisible()`	Virtual function called when the indicator becomes invisible.	(To be overridden in derived classes)

Example of Deriving `IndicatorRenderer` and Overriding Virtual Functions

The `IndicatorRenderer` class includes virtual functions (`protected virtual`) for specific events (entering/exiting the screen, visibility changes). You can override these functions by creating your own custom `IndicatorRenderer` classes, thereby customizing the indicator's behavior.

Example:

C#

```
using UnityEngine;

public class MyCustomIndicator : IndicatorRenderer
{
    // Our own custom indicator class, derived from IndicatorRenderer.

    protected override void OnEnterScreen()
    {
        base.OnEnterScreen(); // Call the base class implementation.
        Debug.Log(gameObject.name + " indicator entered the screen!");
        // Add your custom logic here for when it enters the screen.
        // For example, start an animation or play a sound.
    }

    protected override void OnExitScreen()
    {
        base.OnExitScreen(); // Call the base class implementation.
        Debug.Log(gameObject.name + " indicator exited the screen!");
        // Add your custom logic here for when it exits the screen.
    }

    protected override void OnVisible()
    {
        base.OnVisible(); // Call the base class implementation.
        Debug.Log(gameObject.name + " indicator became visible!");
        // Add your custom logic here for when the indicator becomes visible.
    }

    protected override void OnInvisible()
    {
        base.OnInvisible(); // Call the base class implementation.
        Debug.Log(gameObject.name + " indicator became invisible!");
        // Add your custom logic here for when the indicator becomes invisible.
    }

    // Other custom logic or functions can be added
}
```

After creating this `MyCustomIndicator` class, you can use it by assigning a prefab containing this script in your `IndicatorProfile` for the `Custom` indicator type or as a specific `CustomIndicatorEntry`.

Contact

If you have any questions, feedback, or require support, please do not hesitate to contact us.

samlog.182@outlook.com