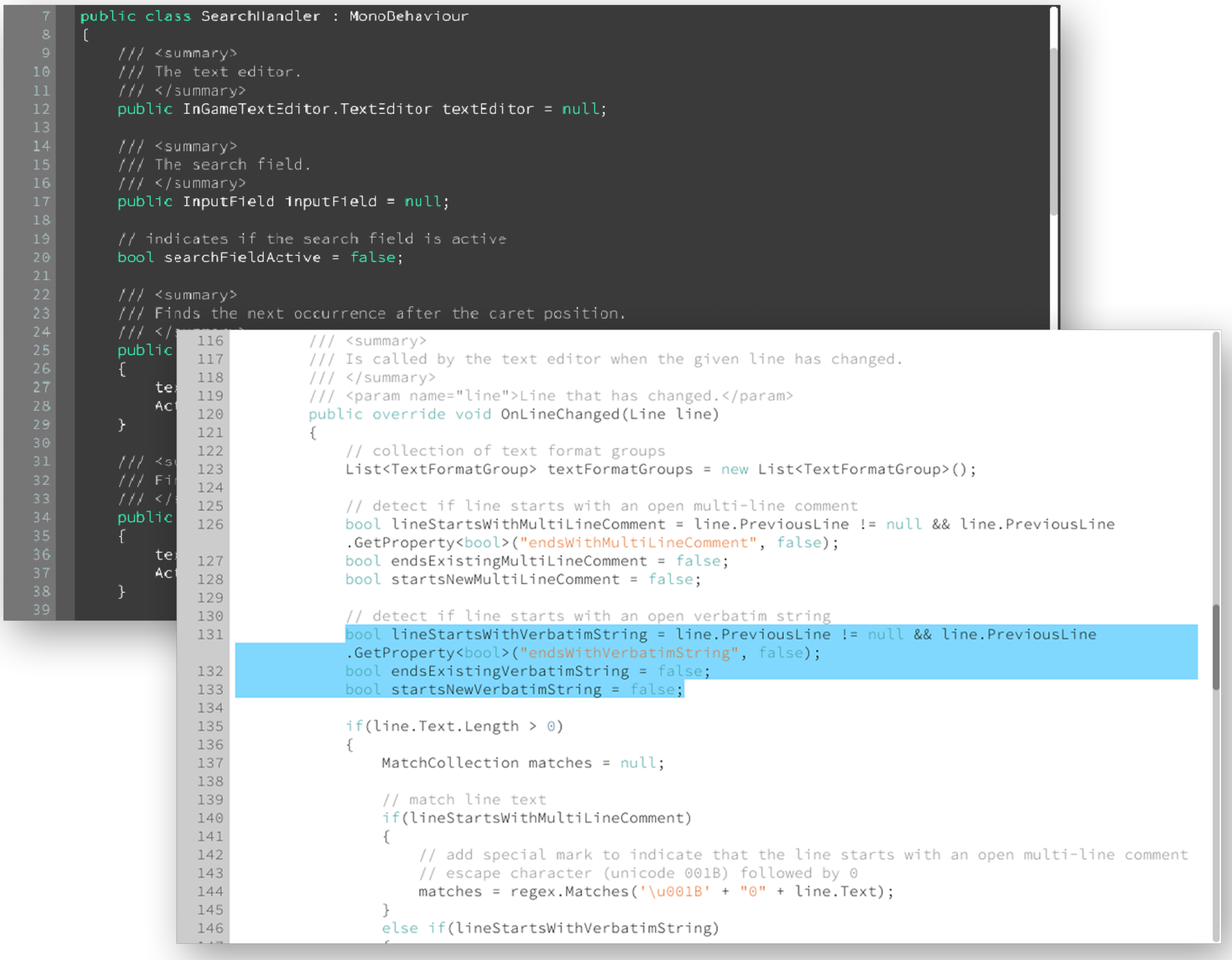
In-Game Text Editor v1.0.4

User Manual



```
public class SearchHandler : MonoBehaviour
{
    /// <summary>
    /// The text editor.
    /// </summary>
    public InGameTextEditor.TextEditor textEditor = null;

    /// <summary>
    /// The search field.
    /// </summary>
    public InputField inputField = null;

    // indicates if the search field is active
    bool searchFieldActive = false;

    /// <summary>
    /// Finds the next occurrence after the caret position.
    /// </
    public
    {
        te
        Ac
    }

    /// <s
    /// Fi
    /// </
    public
    {
        te
        Ac
    }
```

```
        /// <summary>
        /// Is called by the text editor when the given line has changed.
        /// </summary>
        /// <param name="line">Line that has changed.</param>
        public override void OnLineChanged(Line line)
        {
            // collection of text format groups
            List<TextFormatGroup> textFormatGroups = new List<TextFormatGroup>();

            // detect if line starts with an open multi-line comment
            bool lineStartsWithMultiLineComment = line.PreviousLine != null && line.PreviousLine
            .GetProperty<bool>("endsWithMultiLineComment", false);
            bool endsExistingMultiLineComment = false;
            bool startsNewMultiLineComment = false;

            // detect if line starts with an open verbatim string
            bool lineStartsWithVerbatimString = line.PreviousLine != null && line.PreviousLine
            .GetProperty<bool>("endsWithVerbatimString", false);
            bool endsExistingVerbatimString = false;
            bool startsNewVerbatimString = false;

            if(line.Text.Length > 0)
            {
                MatchCollection matches = null;

                // match line text
                if(lineStartsWithMultiLineComment)
                {
                    // add special mark to indicate that the line starts with an open multi-line comment
                    // escape character (unicode 001B) followed by 0
                    matches = regex.Matches('\u001B' + "0" + line.Text);
                }
                else if(lineStartsWithVerbatimString)
```

TABLE OF CONTENTS

1. GETTING STARTED

1.1. Installation

Simply place the InGameTextEditor folder anywhere within your Assets folder.

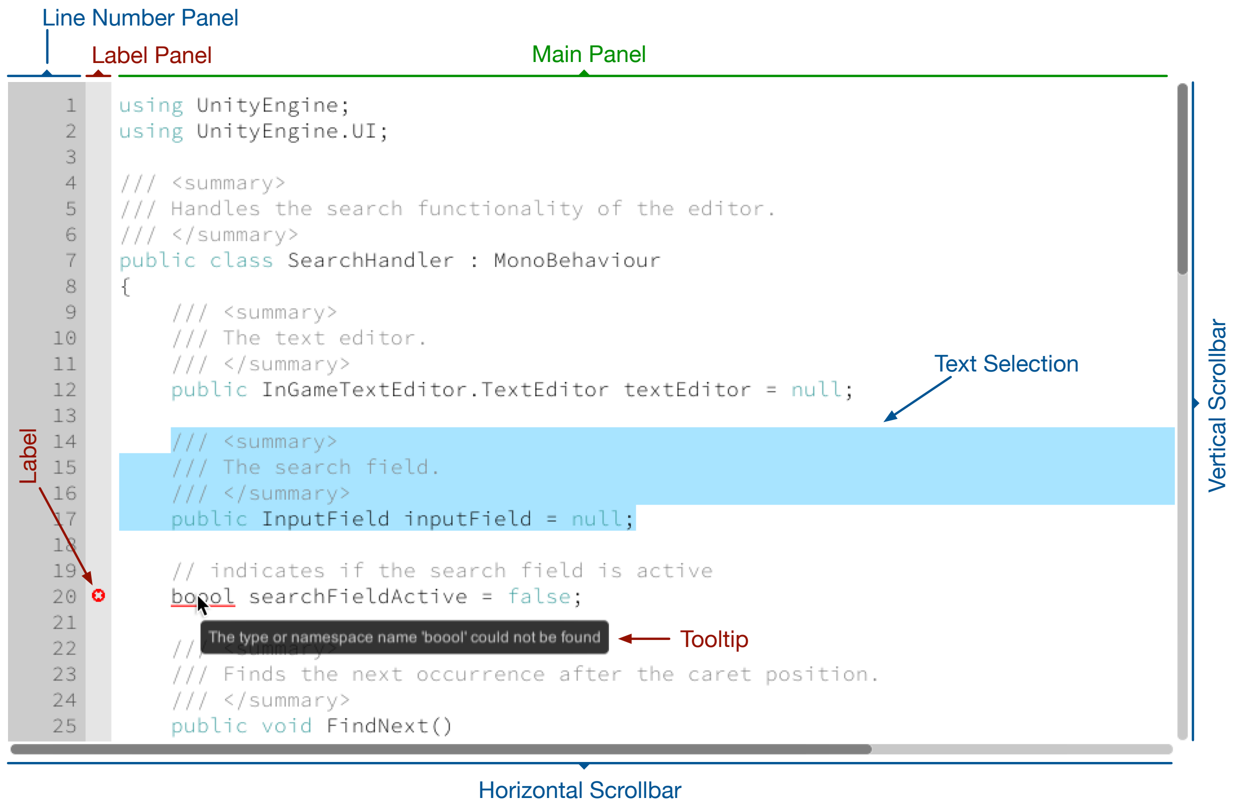
1.2. Adding a text editor to your scene

Your scene needs to have a canvas in which the text editor can be displayed. If you haven't done so already, simply add one from Unity's menu: GameObject | UI | Canvas.

Now drag the TextEditor prefab (Prefabs / TextEditor) to the Canvas.

You now have a text editor in your scene. By default, it covers up the entire canvas. Since it is a normal UI element, you can adjust the size to your requirements.

The example scene 'Simple Text Editor' (inside the Examples / Scenes folder) shows a minimal example of a scene containing a text editor.



```
using UnityEngine;
using UnityEngine.UI;

/// <summary>
/// Handles the search functionality of the editor.
/// </summary>
public class SearchHandler : MonoBehaviour
{
    /// <summary>
    /// The text editor.
    /// </summary>
    public InGameTextEditor.TextEditor textEditor = null;

    /// <summary>
    /// The search field.
    /// </summary>
    public InputField inputField = null;

    // indicates if the search field is active
    boool searchFieldActive = false;

    /// <summary>
    /// Finds the next occurrence after the caret position.
    /// </summary>
    public void FindNext()
```

The type or namespace name 'boool' could not be found

Figure 1: The different components of InGameTextEditor.

1.3. Tweaking the settings

Let's have a look at the settings of the text editor you just added to your scene.

The **Text** section contains all settings related to the presentation of the text inside the editor.

Font	The font used for displaying the text in the content panel as well as the line numbers. This must be a monospaced font. If you select a non-monospaced font, you will get an error message and inconsistent behavior when trying to select text in the editor. InGameTextEditor comes bundled with SourceCodePro, a monospaced font created by Adobe and released under the SIL Open Font License. The font and the license file can be found inside the Fonts folder.
Font Size	The font size used for text and line numbers. It corresponds to the height of a character in pixels.
Line Spacing	Line spacing in text and line numbers. Must be 1.0 or greater.
Main Font Style	The font style (normal, bold, italic, bold and italic) used for the text
Main Font Color	The text color.
Main Background Color	The background color of the text field.
Main Margin (Left, Right, Top, Bottom)	The margin (in pixels) of the text field. Increase these values to have more space between the text and the border of the text field.
Default Text	A default text that is displayed in the editor when the game starts.
Wrap Lines	If this option is active, the text lines will wrap when they do not fit in the text field. If this option is off, the text field needs to scroll left and right to display the entire line.
Indent New Lines	If active, new lines will be indented by the same amount of white spaces or tabs as the previous line.

ⓘ Most of these settings are reflected in the preview of the editor (i.e. when not in play mode).

The **Line Numbers** section controls the looks of the line numbers.

Show Line Numbers	Controls if line numbers are shown. If this option is inactive, the following options have no effect.
Line Number Font Style	The font style (normal, bold, italic, bold and italic) used for the line numbers.
Line Number Font Color	The text color of the line numbers.
Line Number Background Color	The background color of the line number panel.
Line Number Min Width	The line number panel's minimum width (in pixels). If more space is required to accommodate all line numbers, the panel width will automatically increase.
Line Number Margin (Left, Right)	The margin (in pixels) of the line number panel.

The **Line Label Icons** section contains settings related to the bar containing the line labels.

Show Line Label Icons	Shows or hides the entire line label bar.
Line Label Icons Background Color	The background color of the line label panel.
Line Label Icons Width	The width of the line label bar.

ⓘ Line labels are scaled to fill the height of the text line.

Use the **Tooltip** section to control the look and feel of the tooltips.

Tooltip Font	The font of the tooltips. As opposed to the main font, this one does not need to be a monospaced.
Tooltip Font Size	The font size of the tooltips.
Tooltip Line Spacing	Line spacing in the tooltip text.
Tooltip Font Style	The font style (normal, bold, italic, bold and italic) used for the tooltips.
Tooltip Font Color	The color of the tooltip text.
Tooltip Background Color	The background color of the tooltips.
Tooltip Margin (Left, Right, Top, Bottom)	The margin (in pixels) of the tooltip text field.
Tooltip Above Cursor	If the tooltip is shown above the cursor, its bottom end will be placed the given number of pixels above the cursor.
Tooltip Below Cursor	If the tooltip is shown below the cursor, its top end will be placed the given number of pixels below the cursor.
Tooltip Delay	The tooltip will start to appear the given time (in seconds) after it has been activated (i.e. the mouse cursor is on a part of text with a tooltip).
Tooltip Fade Duration	The tooltip will fade in over the given duration (in seconds). Set this to 0.0 if the tooltip should appear immediately without

The **Behavior** section controls the general behavior of the text editor.

Disable Input	Activate this option if you want to prevent changes to the editor's content. For example, if you want to present read-only content.
Deactivate On Lost Application Focus	Deactivate the editor when the application loses focus.
Deactivate On Click Outside Of Editor	Deactivate the editor when a mouse click outside of the editor is registered.
Resize Timeout	The editor layout will be updated the given time (in seconds) after the last resize operation. A value of 0.0 results in immediate updating of the editor layout and may, if the editor holds a lot of text, cause longer loading time when resizing the window.

Show Lock Mask	If active, the editor is covered with a transparent gray lock mask when it is busy. This typically only occurs when pasting a large amount of text.

ⓘ If the editor is not active, it will not respond to any mouse or key input.

The **Scrolling** section controls the scrolling behavior of the text editor.

Show Vertical Scrollbar	Shows or hides the vertical scrollbar.
Show Horizontal Scrollbar	Shows or hides the horizontal scrollbar.
Scrollbar Width	The width of the scrollbars.
Vertical Scroll Speed	The vertical scroll speed when using a mouse wheel or trackpad.
Horizontal Scroll Speed	The horizontal scroll speed when using a mouse wheel or trackpad.
Vertical Drag Scroll Speed	The vertical scroll speed when dragging a text selection out of the editor.
Horizontal Drag Scroll Speed	The horizontal scroll speed when dragging a text selection out of the editor.
Invert Vertical Scroll Direction	Inverts the vertical scroll direction.
Invert Horizontal Scroll Direction	Inverts the horizontal scroll direction.

ⓘ To change the look of the scrollbars, directly modify the scrollbar UI elements inside the text editor.

The **Caret** section controls the look of the caret.

Caret Color	The color of the caret.
Caret Width	The caret’s width in pixels.
Caret Blink Rate	The caret blinks the given frequency (in s^{-1}). This value must be 0.1 or greater.

The **Selection** section controls the look and feel of the text selection.

Selection Active Color	The color of text selection when the editor is active.
Selection Inactive Color	The color of text selection when the editor is inactive.
Double Click Interval	The amount of time (in seconds) in which two subsequent mouse clicks are registered as a double click.
Word Delimiters	A list of characters that separate words. When selecting an entire word using a double click, the current word delimited by any of the given characters is selected.

The **Tab Stops** section defines the tab stop settings.

Tab Stop Width	The number of characters that correspond to the width of one
Replace Tabs By Spaces	If this option is active, tabs are replaced by the corresponding number of spaces.

The **History** settings defines the editor's history.

Enable History	If this option is active, the editor keeps track of changes to its content that can be rolled undo (redo operation) or forward (redo operation).
Max History Length	The maximum amount of changes that are kept in the history. If the history is full, the oldest change will be forgotten.

The **Keyboard** section defines the editor's response to keyboard input.

Use Default Keyboard Shortcuts	Controls whether the default keyboard shortcuts (Ctrl+C, Ctrl+V. etc.) are active.
Use Arrow Keys	If activated, the arrow keys can be used to navigate through the text.
Key Repeat Threshold	The delay (in seconds) to trigger repeated key stroke when holding down a key. The value must be 0.1 or greater.
Key Repeat Rate	A key stroke will be repeated with the given rate (in s^{-1}) when the key is held down. This value must be 0.1 or greater.

The **Internal** section is used to link the text editor components to its MonoBehavior script.

Main Panel	The Main Panel transform. This transform holds the Main Content transform and masks everything outside of its boundaries.
Main Content	The Main Content transform. This transform holds all text blocks, all selection blocks, all labels, and the caret. It is moved based on the vertical and horizontal scroll values.
Text Container	The Text Container. This transform holds all text blocks.
Selection Container	The selection container. This transform holds all selection rects.
Label Container	The label container. This transform holds all text labels.
Main Text	The Main text. It holds the default text when in edit mode and is hidden when the application is running.
Line Number Panel	The line number panel. This transform holds the Line Number Content transform and masks everything outside of its
Line Number Content	Line number content transform. This transform holds all line number text blocks. It is moved based on the vertical scroll
Line Number Text	The line number text. It holds the line numbers displayed in edit mode and is hidden when the applications is running.
Line Label Icons Panel	The line label panel. This transform holds the Line Label Icons Container transform and masks everything outside of its boundaries.

Line Label Icons Content	Line label icons content transform. This transform holds all line label icons. It is moved based on the vertical scroll value.
Caret	The Caret game object. It is displayed at the caret position and displayed or hidden based on whether a text selection is active and blinks with the frequency set with Caret Blink Rate.
Vertical Scrollbar	The vertical scrollbar.
Horizontal Scrollbar	The horizontal scrollbar.
Lock Mask	The lock mask shown on top of editor when busy.
Tooltip	The tooltip window.
Tooltip Text	The tooltip text.

⚠ Changing these settings is not recommended as it may break the behavior of the editor.

2. WORKING WITH THE EDITOR

InGameTextEditor has been designed to look and feel like most text editors. The following section explains the main features.

2.1. Placing the caret

The caret (shown as a blinking vertical bar) designates the position where text is entered. The caret's position can be adjusted by left-clicking with the mouse or by using the arrow keys (left, right, up, down).

Using the right arrow key while holding down 'alt' moves the caret to the end of the next word. The left arrow key accordingly moves it to the beginning of the previous word.

2.2. Text selection

Any part of the text can be selected using the mouse by clicking and holding the left mouse key and dragging the mouse. A blue (default settings) marker highlights the selected text.

Double-clicking a word selects the entire word, triple-clicking selects the entire line.

A text selection can also be created or modified by holding down the shift key and moving the caret using the arrow keys.

Invoking the SelectAll() method or using the keyboard shortcut (default: Ctrl+A) selects the entire text.

2.3. Copy and paste

Selected text can be copied to the clipboard and pasted from it. This can be achieved by using the shortcuts (default: Ctrl+C for copying, Ctrl+V for pasting, Ctrl+X for cutting out text).

In addition to using keyboard shortcuts, these functions can also be used by calling the respective methods Copy(), Cut(), and Paste() on the TextEditor script (see section 3.1).

ⓘ	On macOS, the default keybindings use the Cmd key instead of Ctrl.

2.4. Undo and redo

InGameTextEditor keeps track of all modifications to its content. Using the undo function, the most recent change is made undone. Accordingly, redo reapplies the change that has been undone. The default shortcuts are Ctrl+Z for undo and Ctrl+Y for redo.

ⓘ	On macOS, the shortcuts are Cmd+Z for undo and Shift+Cmd+Z for redo.

Like copy and paste, these functions can be triggered directly by calling the Undo() and Redo() methods (see section 3.1).

3. EXTENDING THE EDITOR

3.1. Public methods

The TextEditor script offers various public methods that can be called from Unity UI elements (e.g. buttons) or from a script.

For example, calling the method ‘Copy()’ stores the selected text to the clipboard. Calling ‘Paste()’ inserts the text stored in the clipboard at the caret’s position.

For a list of all editor methods, please refer to the script reference in the Documentation folder.

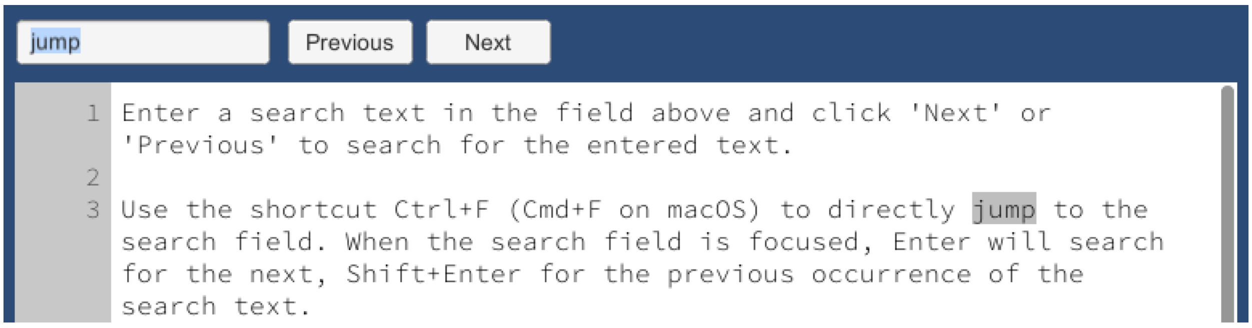
ⓘ Many of these methods accept an optional boolean argument ‘immediately’ (default value: false). Setting immediately = true will execute the respective method immediately and ignore currently queued operations. As a consequence, the executed operation is not recorded in the history and, depending on the complexity, may block the main loop of your game.

The example scene ‘UI Elements’ shows how the copy, cut, paste, undo, and redo methods can be called from UI buttons.

3.2. Text search

InGameTextEditor supports forward and backward search of text. The method ‘Find()’ requires an argument ‘text’ containing the search text and a boolean argument ‘forward’ which determines the search direction (starting from the caret position). If the given search text is found, the text editor selects the search result and automatically scrolls to the appropriate position. If no occurrence of the search text is found, this method does nothing.

The search method can be used in many different ways. The example scene ‘Search’ shows a proposed solution containing a search field and two buttons to search in backward and forward direction. The attached ‘SearchHandler’ script also introduces a shortcut Ctrl+F (Cmd+F on macOS) to directly focus the search field.



```
Enter a search text in the field above and click 'Next' or
'Previous' to search for the entered text.

Use the shortcut Ctrl+F (Cmd+F on macOS) to directly jump to the
search field. When the search field is focused, Enter will search
for the next, Shift+Enter for the previous occurrence of the
search text.
```

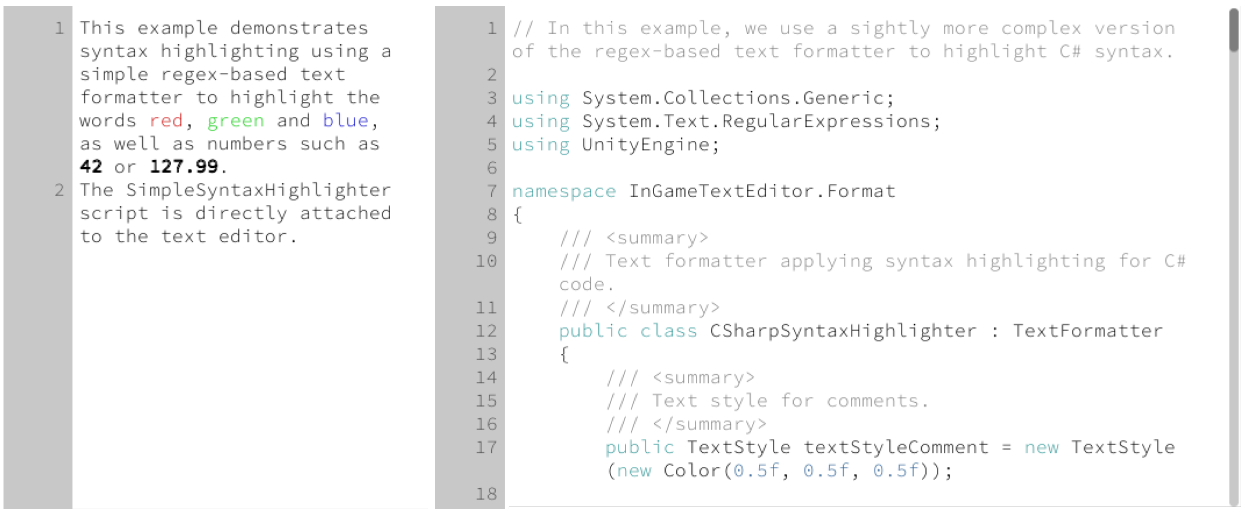
Figure 2: The search example showing a highlighted occurrence of the search text.

3.3. Syntax highlighting

The text editor supports applying different text styles to its content. This can be used to implement syntax highlighting.

The example scenes ‘Simple Syntax Highlighting’ uses the SimpleSyntaxHighlighter script (in Examples / Scripts) to show how a simple syntax highlighter using regular expressions can be implemented.

A more complex version that highlights C# syntax is shown in the ‘C Sharp Syntax Highlighting’ scene.



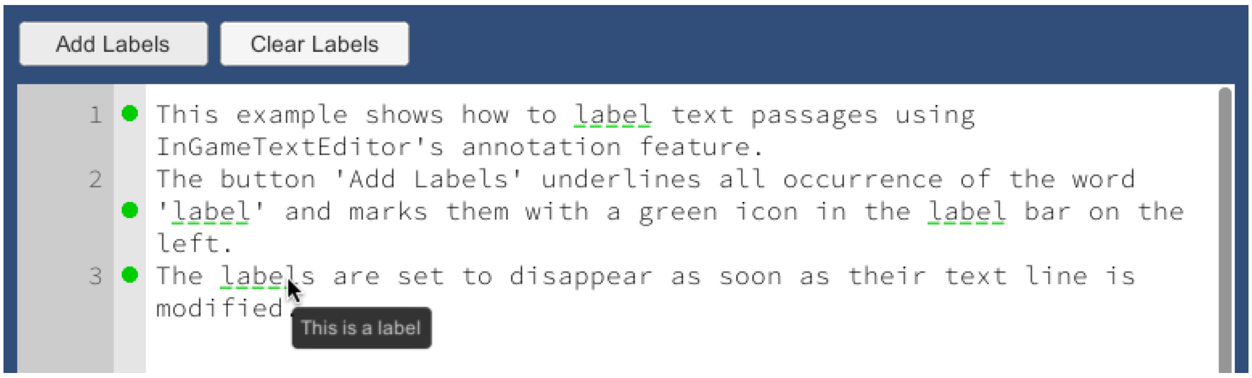
```
This example demonstrates
syntax highlighting using a
simple regex-based text
formatter to highlight the
words red, green and blue,
as well as numbers such as
42 or 127.99.
The SimpleSyntaxHighlighter
script is directly attached
to the text editor.
```

```
// In this example, we use a sightly more complex version
of the regex-based text formatter to highlight C# syntax.

using System.Collections.Generic;
using System.Text.RegularExpressions;
using UnityEngine;

namespace InGameTextEditor.Format
{
    /// <summary>
    /// Text formatter applying syntax highlighting for C#
    code.
    /// </summary>
    public class CSharpSyntaxHighlighter : TextFormatter
    {
        /// <summary>
        /// Text style for comments.
        /// </summary>
        public TextStyle textStyleComment = new TextStyle
        (new Color(0.5f, 0.5f, 0.5f));

```

Figure 3: The example scenes Simple Syntax Highlighting (left) and C Sharp Syntax Highlighting (right).

3.4. Labels and tooltips

InGameTextEditor offers the possibility to annotate parts of the text with labels. These annotations include underlining, highlighting and marking with icons in the label bar. The example scene 'Labels' shows how these labels can be added and removed.



```
This example shows how to label text passages using
InGameTextEditor's annotation feature.
The button 'Add Labels' underlines all occurrence of the word
'label' and marks them with a green icon in the label bar on the
left.
The labels are set to disappear as soon as their text line is
modified.
```

Figure 4: An example of labels with tooltips as demonstrated in the Labels example scene.

4. SUPPORT

If you find a problem with In-Game Text Editor, please do not hesitate to find help on https://ropelato.net or send an email to sandro@ropelato.net.