Fake Terminal - An in-game MS-DOS style PC

Fake Terminal is a script that works with a Text object. It manipulates the strings of the Text object to simulate an old-style PC, where commands are entered by keyboard.

- HOW TO USE:

- Create a world canvas with a Text.

- Add Fake Terminal script as component to the Text.

- HOW TO SET UP IN EDITOR:

- *admin_RequestLogin* is a bool variable. Set it true if you want to request the password when the terminal is activated.

- *admin_Password* is a string. Choose the password. If you leave it blanck the password will be 12345 by deafult.

- *admin_AutoLogOut* is a bool variable. Set it true if you want to log out when you shutdown the terminal. This means you have to insert again the password when you use the terminal again.

- *text_ForceCapsLock*. If it's true all the text will be capitalized.

- *key_TurnOn*. Choose the key you want to use to turn on the terminal.

- *key_UseKeyToShutDown* can be set true if you want to shut down the terminal using a key. Otherwise, you should set a command to write for shuting down the terminal.

- *terminal_PoVOffset* is a Vector3. Use those values to set the position of the camera in front of the terminal, moving it along the Text object axes (rotation is handled automatically).

- *terminal_ActivationTransitionSpeed* is a float variable. A high value increases the camera speed to move from its position, when the key that activates the terminal is pressed, to the position in front of the terminal.

- *terminal_ShutDownTransitionSpeed*. Set a high value to move the camera from the terminal to its starting position.

- *player_CharController*. If your player uses a CharacterController to move, this script automatically deactivates it when you're using the terminal.

- *player_Camera*. It's the main camera.

- *inputs_Help* is an array of strings. The script compares those strings with your input string. The relative function starts if they are the same.

- *inputs_Clear*. This command removes all the lines on the screen.

- *input_ShutDown*. Those string are your command to shut down the terminal (it works only if key_UseKeyToShutDown is false).

- *text_Intro*. Write your custom message to be displayed when the terminal starts.

- *text_Error*. Write the error message to be displayed when the called command is nonexistent.

- *text_Help*. Write the help message to be displayed when you use the relative command.

- *text_AccessNegate*. If the password is wrong, terminal shows this message.

- *text_AccessGranted*. If the password is right, terminal shows this message.

- *text_ShutDownKey*. Print these strings at the end of the help message if key_UseKeyToShutDown is true. Use them to show the key to press for shutting down the terminal.

- *text_ShutDownCommand*. Print these strings at the end of the help message if key_UseKeyToShutDown is false. Use them to show the command for shutting down the terminal.

- *intro_TimeToPrintLine*. It's the time that passes before display the next string of text_Intro array.

- *text_MaxStringsOnScreen*. You have to manually set the maximum lines that can be showed on the fake screen.

- *text_MaxCharsInString*. This integer value sets the maximum number of characters for each line. Use this to avoid text to go off-screen.

- *cursor_Char*. This character is the cursor showed on the screen.

- *cursor_BlinkingTime*. The cursor blinks, so you can choose how fast it has to.

- HOW TO ADD A CUSTOM FUNCTION:

Let's make an example. We want to add a command that allows us to unlock a door through the terminal.

Fist of all, we should have a personal script that controls the door (let's call it DoorController).

```
public DoorController dc;
```

Then we need an array that stores all the input as strings (like /UNLOCK_DOOR).

```
public string[] inputs_UnlockDoor;
```

If we want to print a message when we use the command /UNLOCK_DOOR, we need another array of strings.

```
public string[] text_UnlockDoor;
```

Naturally, we need to create a new custom function to print the message.

```
private void PrintUnlockDoor()
{
	foreach(string line in text_UnlockDoor)
	{
		AddLineToList(line);
	}
}
```

Now it's time to add some lines of code to the function TerminalInput() because we need to compare the command /UNLOCK_DOOR with the keyboard input string. To do this, we add the code inside this if statement: "if(outputText[actualLine].Replace("" + cursor_Char, "").Length > lineIntro.Length)".

```
foreach(string input_Example in inputs_Example)
{
	if(outputText[actualLine].Length >= lineIntro.Length)
	{
		if(outputText[actualLine].Replace("" + cursor_Char, "").Substring(lineIntro.Length) ==
		input_Example)
		{
			printError = false;

			PrintUnlockDoor();

			dc.UnlockDoorFunction();
		}
	}
}
```

And that's all. If somebody writes /UNLOCK_DOOR on the terminal and he presses Enter, will be displayed a message and the placeholder function UnlockDoorFunction() of our placeholder script DoorController will be called.

- CONTACT:

If you have any question or need support send a mail to procedural.imagination@gmail.com.