RflySim3D Console Command Interface Experiment Principles

3. File Directory

Example directory:
[Installation Directory]\RflySimAPIs\3.RflySim3DUE\0.ApiExps\e2_CommandAPI

No.	Experiment Name	Introduction	File location
1	3D Scene Interaction Interface: RflySim3D Console Command Interface Experiment	Method to control different objects in the scene in real time through the RflySim3D console command interface.	1.command\Readme.pdf
2	3D Scene Interaction Interface: RflySim3D Built-in Sc	List the 3D models and scenes that the platform can select, and how t	2.enumerate\Readme.pdf

No.	Experiment Name	Introduction	File location
	ene Model Tour Experiment	o use the exported enumeration data.	

General Description

Needs of Interaction Design

The origin of console command interaction design can be traced back to early computer systems, when graphical user interfaces (GUIs) were not yet popular, and users needed to interact with the system through text commands. With the development of technology, GUI has gradually become the mainstream user interface, but console commands still retain some advantages, such as:

-
Flexibility: Console commands can directly modify and query various parameters and states of the system without using graphical elements such as menus or buttons.

- Efficiency: Console commands can execute multiple operations at once without needing to click or drag the mouse or other input devices multiple times.

-
Compatibility: Console commands can run on different platforms and devices without considering factors such as screen resolution, color depth, and input methods.

Therefore, the design of console commands is widely used in games and simulation fields as a development and debugging tool that complements the user interface.

Console commands in Unreal Engine are powerful tools that allow users to control, debug, and optimize games or applications at runtime. Console commands can be entered via the keyboard or executed by scripts, and can be easily viewed, added, deleted, and modified, and used flexibly according to different needs and scenarios. The main applications include the following aspects:

-
Debugging and Testing: Console commands can help developers check and modify

the states of various parts of the game or application at runtime, such as physics, rendering, sound, input, network, etc. Through console commands, you can quickly locate and solve problems or simulate different situations for testing.

-
Optimization and Analysis: Console commands can display and adjust the performance parameters of the game or application, such as frame rate, memory usage, CPU and GPU usage, the number of objects, etc. Through console commands, you can monitor and optimize the running efficiency of games or applications, or analyze their resource consumption.

-
Interaction and Control: Console commands allow users to change the behavior and performance of a game or application at runtime, such as viewing angle, screen effects, operation mode, difficulty, etc. Through console commands, you can increase the playability and flexibility of games or applications, or adapt to different needs and preferences.

-
Expansion and Creation: Console commands can implement additional functions and effects, such as creating and destroying objects, playing animations and sounds, executing scripts and commands, etc. Through console commands, you can expand and create the content and experience of games or applications, or achieve specific simulation purposes.

Workflow of Unreal Engine Console

First, to implement console commands, there must be a console window, which is the interface where users can input and view commands. In Unreal Engine, the console window is defined by the SConsoleWidget class, which is a custom control inherited from SCompoundWidget. It contains a text input box, a text output box, and a candidate list. Users can open or close the console window by pressing specific keys, such as `~` or Tab, and can also browse historical or candidate commands through the up and down arrows.

Second, to implement console commands, a console manager is needed. This is the core class responsible for registering, storing, finding, and executing commands. In Unreal Engine, the console manager is defined by the IConsoleManager interface, which is a singleton class whose instance can be obtained by GEngine->GetConsoleManager(). The console manager maintains a TMap variable to store all

registered console objects, where the key is the name of the console object and the value is a pointer to the console object. The console object is defined by the IConsoleObject interface, which is the base class for all console commands, variables, and aliases, and contains some common properties and methods. Console objects fall into three derived classes: IConsoleCommand, IConsoleVariable, and IConsoleAlias, which correspond to the implementation of console commands, variables, and aliases respectively.

Finally, to implement console commands, a console proxy is needed. This is an auxiliary class responsible for processing the input and output of the console window. In Unreal Engine, the console proxy is defined by the FConsoleManager class, which is a class that implements the IConsoleManager interface while also inheriting the FOutputDevice class. The console proxy overrides the Serialize method of the FOutputDevice class to print information to the text output box of the console window. The console proxy also has an Exec method, which is used to parse and execute the commands in the text input box of the console window. The Exec method first checks whether the input string is empty or starts with a special character, and if it is, it returns directly. It then splits the input string into multiple substrings delimited by spaces or semicolons. For each substring, it calls the ProcessUserInput method, which first attempts to find a console object matching the substring from the console manager. If one is found, it performs the corresponding action according to the type of console object. For example, if the console object is a command, it calls its Execute method. If the console object is a variable, it calls its Set method. If the console object is an alias, it recursively calls the Exec method. If no matching console object is found, it calls the FWorldDelegates::OnConsoleCommand method, which iterates through all world agents and attempts to execute the substring internally. If it still fails to execute the substring, an error message is printed.

Console Variables of Unreal Engine[4]

Console variables
can be used to save some state information, which can be modified or viewed through the console. In-game consoles also support autocomplete and enumeration of console commands and console variables registered by the console manager (the console command Help or DumpConsoleVariables). Therefore, avoid using the old Exec interface. The console manager at the intermediate point will control everything and more (such as user input history).

Priority is a 32-bit integer that identifies the relative order in which console variables from different sources are set. The higher the priority, the later the setting was applied, and thus the more likely it is to overwrite previous settings. For example, if a console variable is set to 10 by consolevariables.ini and then 20 on the command line, the final value will be 20 because the priority of the command line is higher than that of consolevariables.ini. Priorities are established based on the following principles:

-
The initial value of the console variable (specified by the constructor) should be the lowest priority so that settings from other sources can take effect.

-
Predefined configuration files (such as Scalability.ini, SystemSettings.ini, DeviceProfiles.ini) should have a medium priority so that users can adjust them according to different platforms and performance needs.

-
User-defined configuration files (such as consolevariables.ini, ProjectSettings.ini) should have a higher priority so that users can override predefined settings and achieve more personalized control.

-
Command line parameters should have the highest priority so that users can temporarily modify certain settings when starting the game or test the effect of different parameters.

-
Settings in code (such as via IConsoleManager::SetConsoleVariable() called through code) should have a slightly lower priority than the command line, in order to enforce modification of certain settings under special circumstances, while also allowing users to override via the command line.

-
Console input (whether from the console window in the editor or the console key in the game) should have the highest priority, so that users can change certain settings in real time while tracking or restoring defaults.

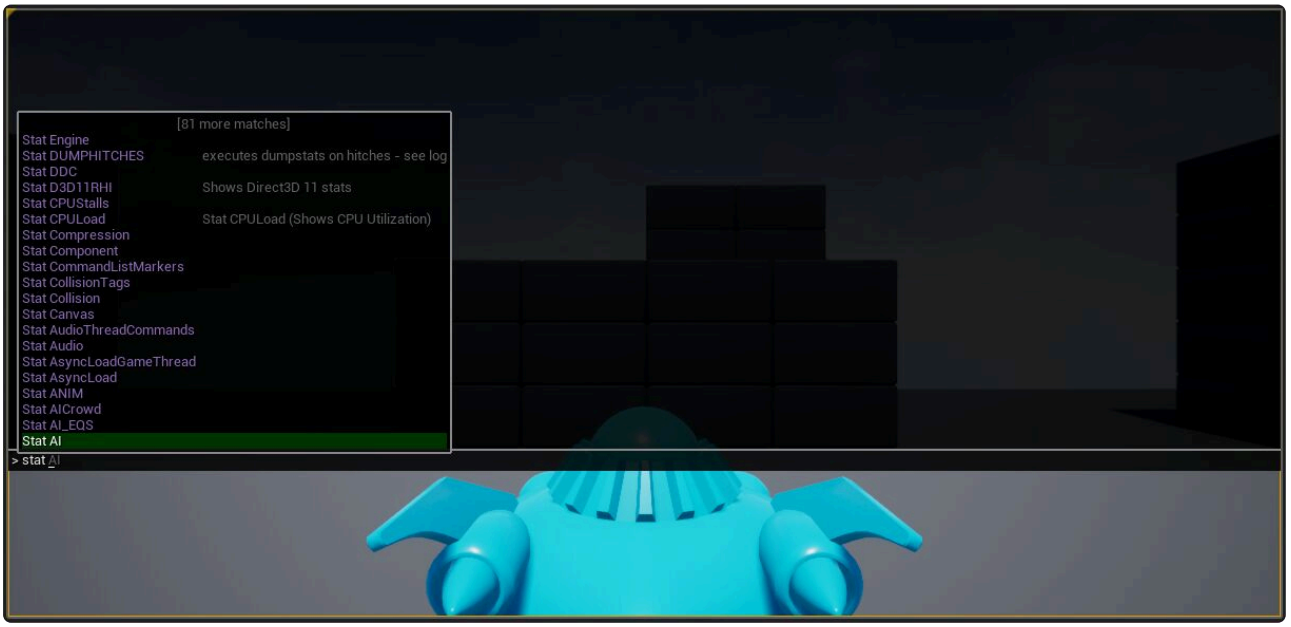
Console Commands of Unreal Engine

A console command is a string sent to the engine, usually entered by the user in the in-game console. The engine can recognize and respond to it in some way (e.g., console/log response, changing internal state). Console variables are special console commands used to store values, which can be modified or viewed through the console.

IConsoleManager is an interface that provides access to and manipulation of console commands. IConsoleManager provides multiple methods for creating, registering, unregistering, tracking, and executing console commands. For example, the RegisterConsoleCommand method can create a console command, specifying its name, description, and delegate function, which is used to define what should be done when the command is executed. The FindConsoleVariable method returns a pointer to a console variable by its name. The ProcessUserConsoleInput method accepts a string and executes it as a console command. To use IConsoleManager, you need to include the "ConsoleManager.h" header file and get a reference to the global console manager object by calling IConsoleManager::Get(). Console manager methods can then be called on this object.

Console Window of Unreal Scene Editor[3]

PIE console (Play In Editor Console)
is an in-game console that allows you to enter commands and display performance data, enable or disable engine features, or perform other actions.



RflySim3D Custom Global Functions

RflySim3D has built-in global functions, which are some functional commands that can be input in the console terminal or accessed through code calls and used to implement various operations related to RflySim3D. The following is a brief explanation using RflyReqObjData as an example:

- RflyReqObjData(int opFlag, FString objName, FString colorStr): Request data of a certain object in the scene. opFlag is an integer representing the requested operation type, objName is a string representing the object name, and colorStr is a string representing the object color. It can be used to acquire position, attitude, velocity, acceleration, and other information of objects, and can be marked with different colors. Supported operations include:

- 0: Request the position of the object (unit: m)

- 1: Request the attitude of the object (unit: degrees)

- 2: Request the velocity of the object (unit: m/s)

- 3: Request the acceleration of the object (unit: m/s^2)

- 4: Request the angular velocity of the object (unit: rad/s)

- 5: Request the angular acceleration of the object (unit: rad/s^2)

- 6: Cancel request

Implementation of Key Functions

RflySim3D has some built-in global commands, which can complete most capabilities associated with RflySim3D. Press the tilde (~) key in the RflySim3D interface window to open the console terminal, where you can trigger some built-in commands of the UE engine, as well as commands built into the RflySim3D platform itself. These commands are essentially equivalent to global functions.

Among them, RflySim built-in commands provide functionalities to run controls and adjust different scene objects, cameras, maps, etc. They mainly include on-screen tooltip displays, map/perspective switching, model/animation previews and controls, and all kinds of data retracement within scenes.

Commands built into the UE engine are utilized for monitoring and balancing assorted rendering configuration variables to optimize simulation performance. The core concepts of performance optimization here encompass two main points: rendering efficiency and computing capacity. Computing capacity entails reducing the count of CPU calculations and pushing items capable of GPU computation into the GPU to establish parallel computing. Rendering efficiency pertains to lowering the amount of draws produced per frame. Moderating the frame rate, post-processing status, shadow condition, and analogous elements can stabilize the rendering performance in compliance with both picture value and presentation requirements. Meanwhile, you can determine and augment computing capacity performance by varying simulation speed functionality and assessing procedure responses.

- See [1] 5. command-line control interface for details of call techniques for complete RflySim3D console commands
- A specific call procedure outline string for RflySim3D console commands can be located at <Readme.pdf>

Related Documents

1. ....\API.pdf
2. IConsoleManager | Unreal Engine Documentation
3. Play In Editor (PIE) | Unreal Engine 4.26 Documentation (unrealengine.com)
   Console Variables in C++ in Unreal Engine | Unreal Engine 5.4 Documentation | Epi
4. c Developer Community (epicgames.com)
5.

Additional Resources

Official Document: RflySim official document: https://rflysim.com/doc/en/

Community Communication: Join the RflySim technical communication group: 951534390


RFLYSIM