GenericCompositeSystem Documentation

Overview

The GenericCompositeSystem is a generic implementation of the Composite Design Pattern.

It enables the creation and management of hierarchical structures like trees or nested components.

The system provides a way to compose objects into tree structures and work with them as if they were individual objects.

Key Features

- Generic Data Handling: Supports any type of data through generic type T.
- Hierarchy Management: Allows adding, removing, and traversing child components.
- Search Capabilities: Supports depth-first and breadth-first searches.
- Traversal Options: Pre-order, post-order, and level-order traversal.
- JSON Conversion: Easily serialize and deserialize tree structures.

Class Structure

The main class is GenericCompositeSystem<T>, which provides the core functionality.

Methods

- void Add(GenericCompositeSystem<T>): Adds a child to the current component.
- void Remove(GenericCompositeSystem<T>): Removes a child from the current component.
- void SetFinal(bool): Sets whether the component can have children.
- IEnumerable<GenericCompositeSystem<T>> GetChildren(): Returns the direct children of the

component.

- string GetPath(): Returns the hierarchical path from the root to the current component.

- int GetDepth(): Returns the depth of the current component in the tree.

- GenericCompositeSystem<T> Find(Func<GenericCompositeSystem<T>, bool>, SearchType): Finds a component matching a predicate using the specified search type.

- void Traverse(Action<GenericCompositeSystem<T>>, TraversalType): Traverses the tree and performs an action on each component.

Code Examples

Creating a tree:

```
var root = new GenericCompositeSystem<string>("Root");

var child1 = new GenericCompositeSystem<string>("Child 1");

var child2 = new GenericCompositeSystem<string>("Child 2");

root.Add(child1);

root.Add(child2);

var grandChild = new GenericCompositeSystem<string>("Grandchild");

child1.Add(grandChild);
```

Traversing the tree:

```
root.Traverse(node => Console.WriteLine(node.Data), TraversalType.PreOrder);
```

Output (PreOrder):

Root

Child 1

Grandchild

Child 2

Searching the tree:

```
var foundNode = root.Find(node => node.Data == "Grandchild", SearchType.DepthFirst);

Console.WriteLine(foundNode?.GetPath()); // Output: root -> 0.child -> 0.child
```

Converting to/from JSON:

```
var jsonTree = TreeHelper.ToJson(root);

var rootFromJson = TreeHelper.CreateTreeFromJson(jsonTree);
```