Complex System Project Brief

Unity Modular Utility AI

**Introduction**

The Unity Utility AI Modular System is a package designed to simplify the integration of AI systems into Unity projects. The system allows developers to effortlessly add behaviours, customize evaluations, conditions, and call-backs through delegates and Unity events. By supplying control over behaviour evaluation and behaviour events, developers can create AI without extensive knowledge of underlying AI systems.

**The purpose**

The primary purpose of the Unity Utility AI System is to offer a user-friendly package that allows developers with limited AI knowledge to implement AI behaviours into their projects, new or existing. The system relies on delegates and Unity events to provide developers with flexibility in adding behaviour events, evaluations, and conditions. This gives them the ability to design AI behaviours precisely to their project's requirements, without needing in-depth knowledge of the underlying systems. By abstracting the AI systems, developers can focus on other parts of their project without spending significant time implementing AI.

**External Libraries**

The Unity Utility AI Modular System relies only on the in-built Unity systems, there are no external libraries or dependencies used in the package. This ensures compatibility and stability, while simplifying the integration process.

**Modularity**

The system achieves modularity using the Unity inspector. Key aspects of its modularity include tools for generating numerous behaviours directly from the inspector; and automated script generation, creating new a C# script with the Utility AI system and serialized variables for each custom behaviour selected by the developer. Developers can then inherit from this generated script to access and modify behaviour functionality.

**Behaviour Functionalities**

Once the behaviours are generated, functionalities can be accessed and customized through the script inspector. Each behaviour becomes a field in the inspector, providing a convenient interface for modifying various aspects, including:

* *Evaluation Method*: Developers can define a method that calculates a value based on the game situation. This method contributes to the selection of behaviours by providing raw scores within the user-defined value range. This score is used by the AI system to choose which behaviour should be active.
* *Condition methods*: Through the inspector, developers can add methods which return true or false. These are called conditions and they determine when a behaviour should be interrupted or deactivated. This allows for precise control over behaviour transitions based on changing game circumstances. The system comes with an inbuilt timer condition which allows behaviours to end after a certain duration.
* *Behaviour Events*: Behaviour events are the actions or behaviours that occur when a behaviour starts, is active, and ends. By configuring the behaviour events in the inspector, developers can specify the desired actions or effects associated with each behaviour.
* The system also provides customisability of the method used to select certain behaviours. This method takes in an array of weights provided by the AI backend, representing the scores of each behaviour. It returns an integer value representing an index of one of those values, to be chosen as the next behaviour. The system provides three inbuilt methods that can be used, a weighted random selection, highest selection, or lowest selection; or the developer can make their own.

**Mathematical Operations**

Mathematical operations are crucial for calculating new behaviours within the system. By default, behaviour selection uses weighted randomization based on scores. The evaluation of scores has two types of values: a user-defined value range (a raw value) and a normalized value between 0 and 1. Evaluation methods should return values within the user-defined range, which serve as a behaviour’s raw score. Later during Utility AI calculations, these raw scores are remapped to a 0-1 range, ensuring unbiased behaviour selection across all ranges.

**Advanced Algorithms and Systems**

The Unity editor lacks an inbuilt system for displaying delegates in the inspector. Although Unity events are supported, they do not allow for return types, which are necessary for evaluators, conditions, and behaviour selectors. To address this limitation, a custom Unity editor drawer system is implemented. Leveraging a container class and reflection, the editor can display all methods of an object in a dropdown menu, allowing for the creation of delegates by using inspector exposed method names.

**Integration into a New Unity Project**

Integrating the Unity Utility AI Modular System into a new Unity project involves the following steps:

* Import the Modular Utility AI package into the Unity project.
* Create a new script that inherits from the UtilityAI class.
* Attach the newly created script to an object, such as an enemy, in the Unity editor.
* In the inspector, add desired behaviours to the behaviours list (behaviours can be added, updated, or removed at any point in the future).
* Press the "Generate AI Instance" button to create a new Utility AI behind the scenes. Your created script should now inherit from this generated script and include an interface with AIAwake, AIStart, and AIUpdate methods.
* Each behaviour becomes a field in the script inspector, exposing functionalities such as evaluation methods, conditions, and active events for editing.

**Conclusion**

The Unity Utility AI Modular System offers a simple and accessible solution for implementing AI systems in Unity projects. By providing an easy-to-use package, which provides tools for creating behaviours, and customizable call-backs, evaluations, and conditions, without the need for extensive AI knowledge. Since the system does not rely on external libraries, it provides compatibility with a wider range of Unity projects. With the Unity Utility AI Modular System, developers can seamlessly integrate AI into their Unity projects.