Utility AI System

Modular AI Unity package

Benjamin Wharton

Utility AI is a form of artificial intelligence that enables AI agents to make rational choices by considering the desirability of different behaviours. It involves calculating values for certain actions based on how “desired” they are, which are used to choose the behaviours that are the most desired by the AI. The Unity Utility AI Modular System is a package designed to simplify the integration of this Utility AI system 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, AI can be developed and implemented within a wide range of project types without extensive knowledge of the 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 and Dependencies**

The Utility AI Modular System relies only on Unity engine and its in-built systems. There are no external libraries, dependencies, or Unity packages used in the AI system. The system is developed for Unity version 2021.3.13f1, however it should be functional within newer and some earlier versions.

# **Modularity**

The system achieves modularity using the Unity inspector and dynamic code generation. Key aspects of its modularity include tools for generating numerous behaviours directly from the inspector; and automated script generation, which involves creating new a C# script that utilises the Utility AI system and includes serialized variables for each custom behaviour selected by the developer, allowing for easy user access to behaviours within both the code and inspector. The generated script can be inherited from custom scripts, allowing expansion upon and use of the backend systems.

The behaviours inside of a generated Utility AI class host functionalities that can be accessed and customized through the code or Unity inspector. Many of the functionalities in each behaviour are delegate methods which can be replaced by any user programmed methods, given that the method has the correct parameters and return type. Each behaviour is represented as a variable and shown as a field in the inspector, providing a convenient interface for modifying these various aspects. The functionalities that can be changed include:

* *Evaluation Method*: A delegate method that can be assigned to which calculates a value based on the game situation. This method is used by the AI system to calculate the normalised score of a behaviour by using the returned raw score that is within the user-defined value range. This score is used to choose which behaviour should be active.
* *Condition methods*: A list of delegate methods that can be assigned to which return true or false. These are called conditions and they determine whether a behaviour should be interrupted or deactivated, and a new behaviour should be found. 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 Unity Events that can be assigned to which are the actions that occur when a behaviour *Starts*, *Is Active*, and *Ends*. By configuring the behaviour events, desired actions or effects associated with each behaviour can be called.
* *Behaviour Selector*: The system also supplies customisability of the delegate method used to select certain behaviours. This method takes in an array of weights provided by the AI backend, being the normalised scores of each behaviour. It must return an integer value representing an index of one of those values, to be chosen as the next behaviour. The system supplies three inbuilt methods that can be used, a weighted random selection, highest selection, or lowest selection; or a custom method can be programmed and assigned to the delegate.

# **Mathematical Operations and Advanced Algorithms & Systems**

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 raw-value within a user-defined range which is returned by evaluation methods, and a normalized value between 0 and 1 which is calculated by the AI system.

The use of normalized values aims to ensure that behaviour evaluations are consistently treated within the same range during AI calculations, preventing skewed results. For instance, consider two scores: one is 5 within a range of 0-10, and the other is 22.5 within a range of 15-30. After normalisation, both scores would be 0.5, which shows that they are 50% of their respective evaluation ranges. This normalisation allows all scores to be treated as percentages relative to their individual ranges rather than arbitrary scores. By doing this, the behaviour selection function can work with scores that uniformly represent a 0-1 range, ensuring fair and consistent comparisons amongst different behaviours.

The second key feature of the system is the use of the Unity inspector for exposing behaviour functionalities. Many of these functionalities utilise C# delegates, however, 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 the backend AI to receive proper response from user supplied evaluators, conditions, and behaviour selectors. To address this limitation, a custom Unity editor drawer system is to be 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 is compatible and seamlessly integrate able with a wide range of Unity projects.