GOAP NPC PLUGIN MANUAL

Diego Romero.

Mario Sánchez.

José Manuel Sierra.

INDEX

Introduction

GOAP NPC plugin has two main classes from which it is possible to generate c ++ classes or Blueprints that inherit from them, being possible to use the functions provided or implement new functionality. These two classes are GOAPController and GOAPAction. In this guide we will focus on the use of the plugin using Blueprints.

Version notes here.

Instalation

The plugin can be obtained from **UE4 Marketplace** or downloading it from our **GitHub repository**. If you install it from the Marketplace, you can skip this section.

To install the project by downloading it from github, you must follow the following steps depending on whether you want to install to a project or to Unreal Engine.

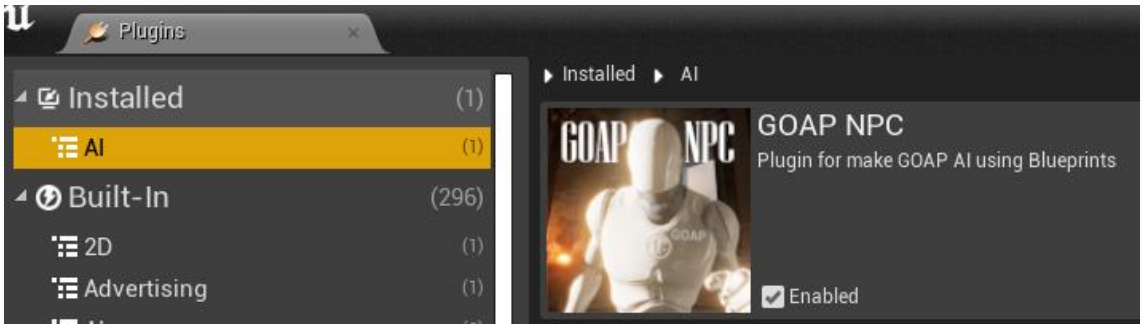
- **Install to project**

  1. Go to project folder which contains the **.uproject** file.
  2. Create a folder called **Plugins**.
  3. Copy **GOAP** folder, downloaded from GitHub, into the **Plugins** folder.

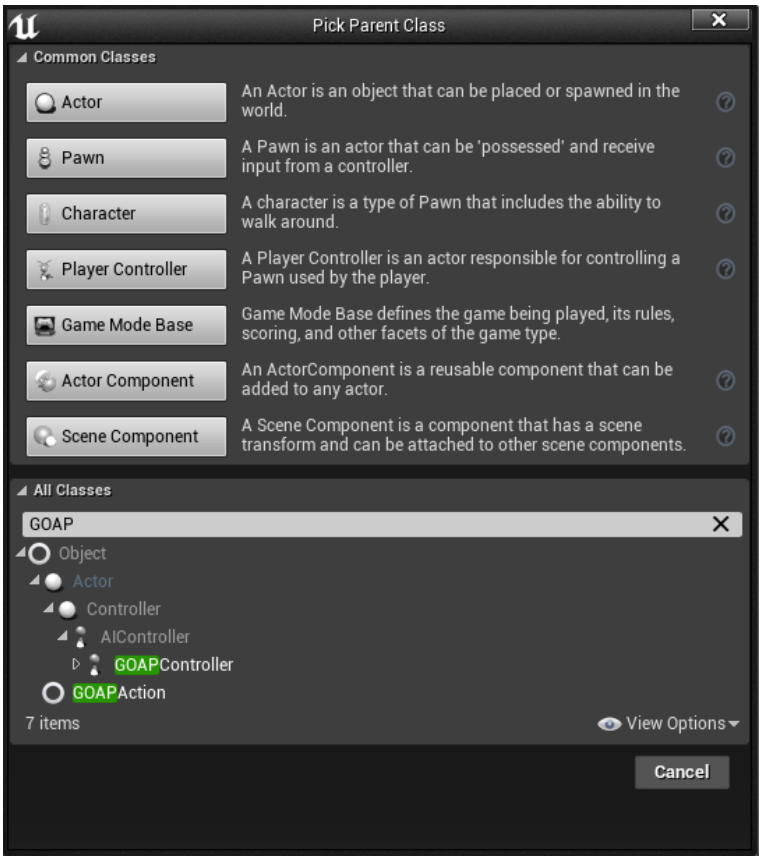
- **Install to Unreal Engine**

  1. Go to Unreal Engine plugins folder, located into **Engine** folder.
  2. Copy **GOAP** folder, downloaded from GitHub, into the **Plugins** folder.

It is necessary to enable the plugin in the UE4 editor, accessing it from the toolbar, Edit-> Plugins.

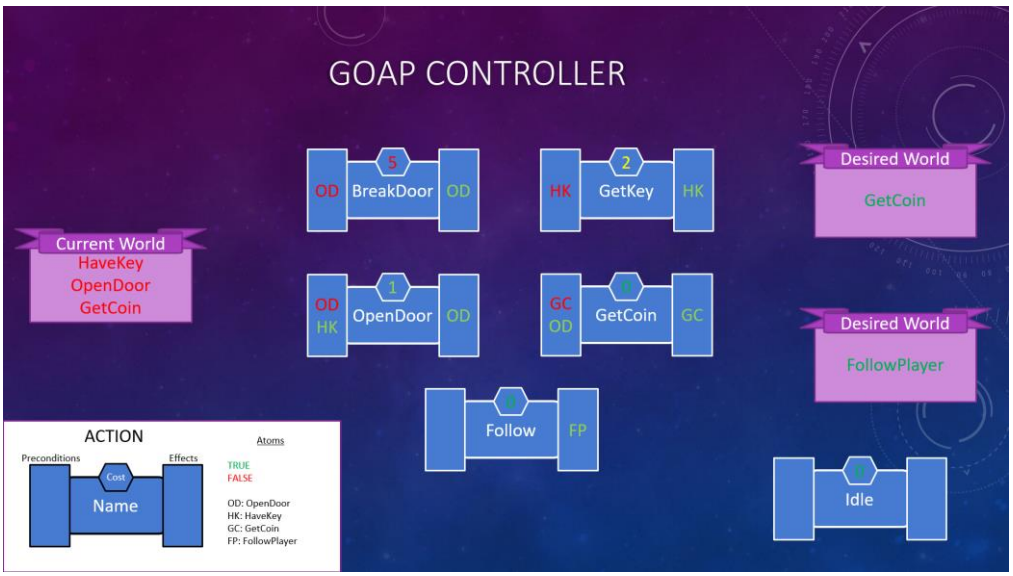


Once this is done it is as simple as create Blueprints and select GOAPController or GOAPAction as parent classes, depending on the type of Blueprint you want to implement.



There is an example project in our [GitHub repository](), this example shows the use of the plugin, you need to have the plugin installed to be able to open it.

The following image represents the GOAPController content and the different GOAPActions of the demo.



Demo's content sketch.

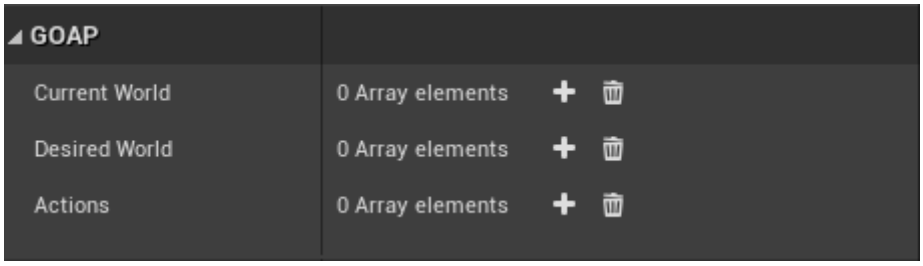
GOAPController

This class inherits from AIController, it is a basic controller with added functionality to be able to use GOAP.

Attributes

- Current world state (Starter world).
- Desired world state (Goals).
- Actions.

These attributes can be found under the GOAP tag in the Blueprint's details tab.



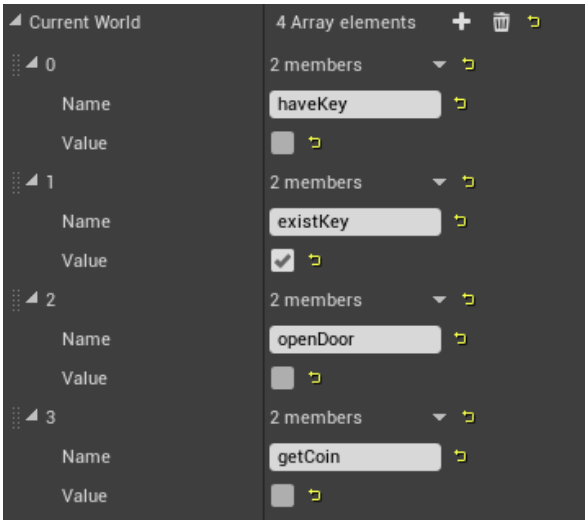
Current world state

Array of pairs formed by a String and a Boolean. onsiste en un Array de pares formados por un String y un Booleano. Predicates that define characteristics of the world that we want our AI to consider.

These predicates are private, that is, each character controlled by a GOAPController will manage them by itself, it will not influence the worlds of the other characters. If you want them to be public, the developer can implement global or hybrid worlds since they can be modified using set() functions.

Example:

Current world state: the character doesn't have a key nor has he got the coin, exist a key and the door is closed.

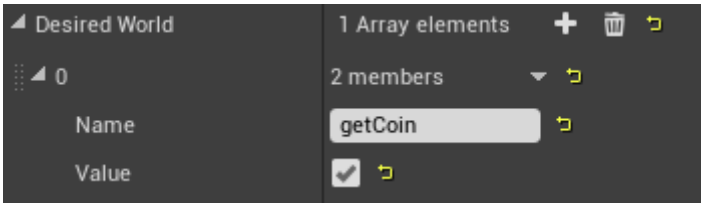


Desired world state (Goals)

Like current world state, it has the predicates that make up the world state when the goal is reached. It is not necessary to deny all the predicates of the initial world, nor is it necessary for it to exist in the initial world, it is enough to declare how we want the world state to be once the goal is accomplished.

Example:

> The desired world state is to have got the coin, since the goal of AI is to get the coin. The rest of the world doesn't matter to us.

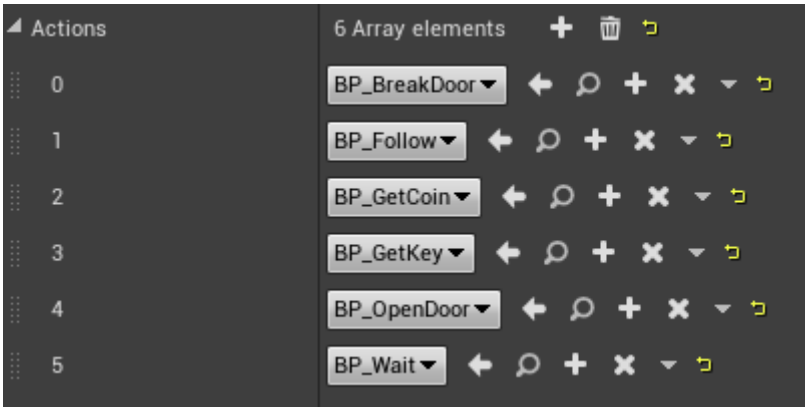


Actions

Array of Bluesprints that inherit from GOAPAction. List of actions that the character can perform.

Example:

> The character can: break doors, follow the player, get coins, get keys, open doors and wait (idle).

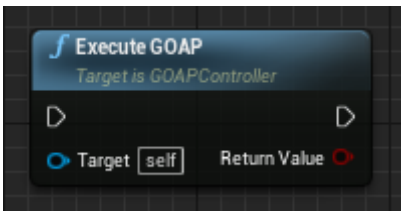


Functions

In addition to the functions that AIController has, GOAPController adds three main functions using GOAPController as input parameter:

ExecuteGOAP

Function that generates a plan and executes the first action of that plan, modifying the world state, if it is called consecutively it will generate new plans according to the circumstances.

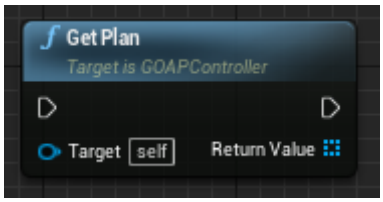


GeneratePlan

Generates a plan according to the world states.

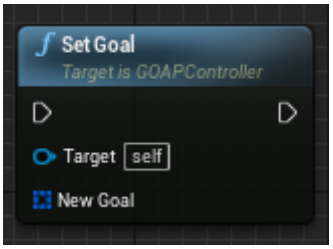
GetPlan

Returns the last generated plan.



SetGoal

Sets a new desired world state.



SetCurrentWorld

Sets the current world state of the AI, this function can also be used to change the current world state.

UpdateGoal

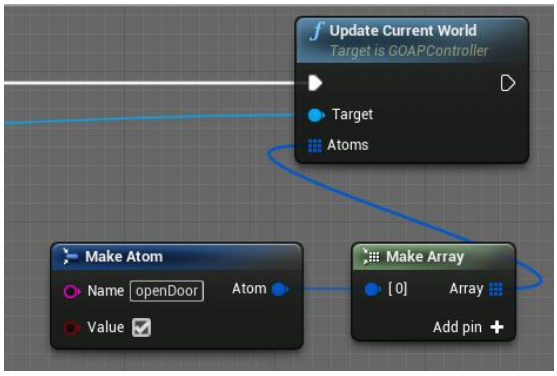
Adds or modifies atoms from the desired world state. Atoms already existing in the desired world state are modified by new input values.

UpdateCurentWorld

Adds or modifies atoms from the current world state. Atoms already existing in the current world state are modified by new input values.

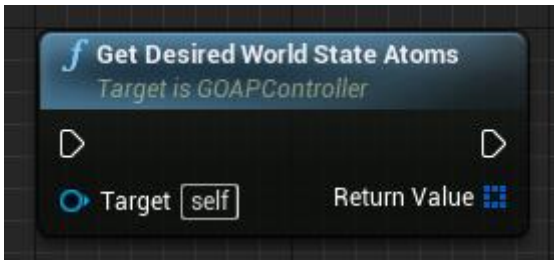
Example:

Change an atom of the current world state to open the door.



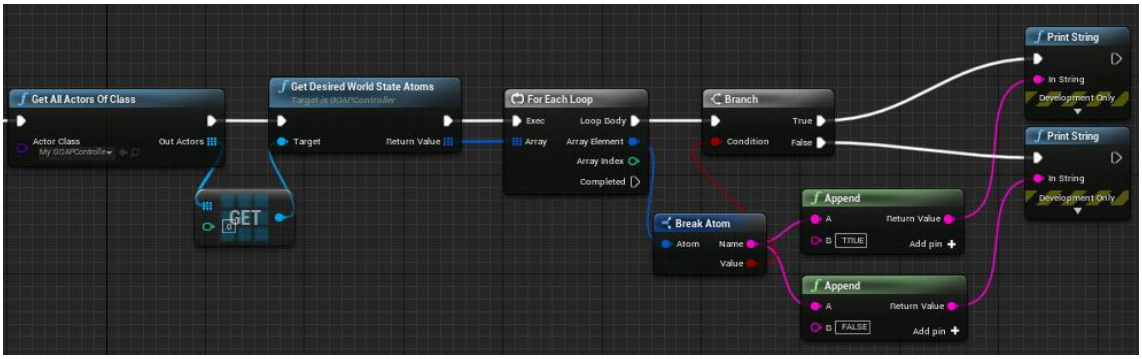
GetDesiredWorldStateAtoms

Returns the desired world state atoms.



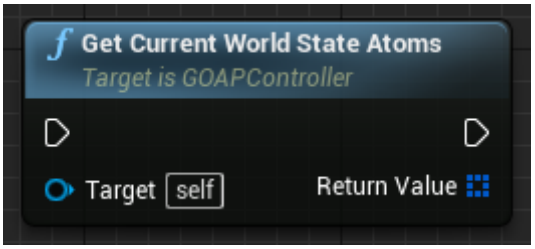
Example:

Prints the atoms that make up the current world state, using “Name VALUE” format.



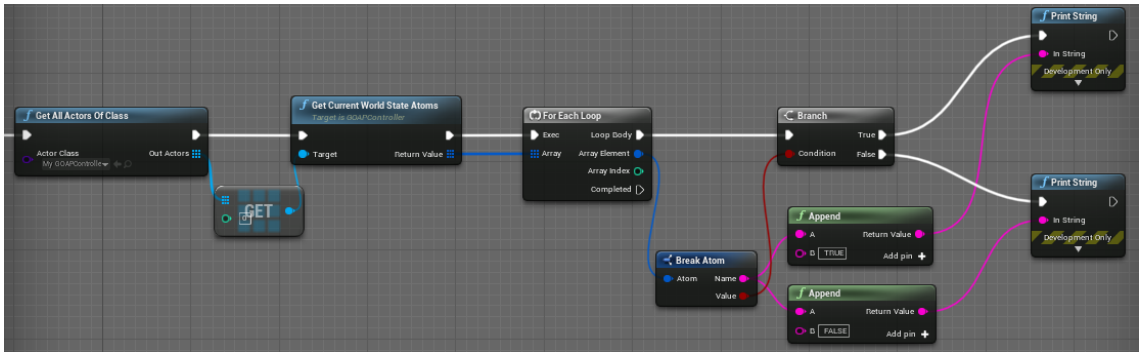
GetCurrentWorldStateAtoms

Returns the current world state atoms.



Example:

Prints the atoms that make up the desired world state, using “Name VALUE” format.

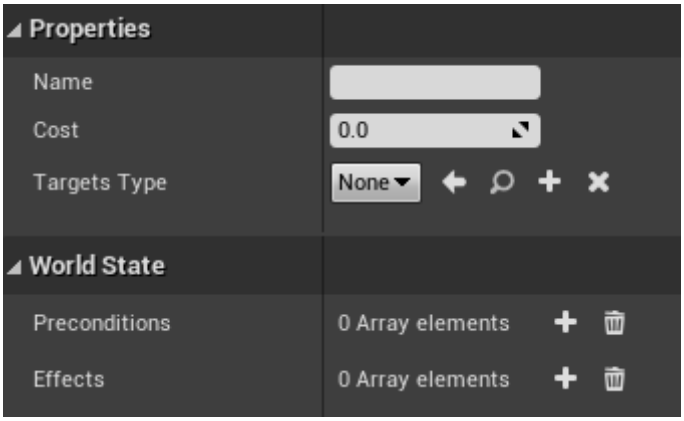


GOAPAction

This class contains the characteristics and logic of an action.

Attributes

- Name
- Cost
- Targets type
- Preconditions
- Effects



Name

String that defines the value of the action, is irrelevant to the plan, it can be empty.

Cost

Float with the value of the action's cost, the algorithm of the plan will take into account the cost when generating it.

Targets type

Type of actor in case the action we define must be performed on a target, it may not be declared if the action does not need an actor as target, for example a move to action.

Example:

Action: Get a key

Target: Key

Targets that meet the type can be obtained by calling the getTargetsList function.

It is also possible to declare a single target using setTarget () and obtain it using getTarget ().

Preconditions

Array of pairs formed by a String and a Boolean. Predicates that must be fulfilled in the world state for the action to be performed.

Effects

Array of pairs formed by a String and a Boolean. Predicates that will be established in the world state once the action has been carried out, if they already existed, they will be modified; if they did not exist, they will be added.

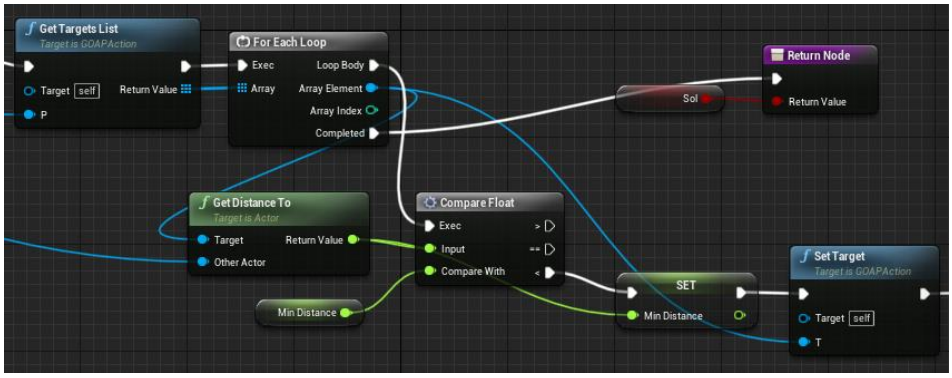
Functions

GetTargetsList

Receives a pawn and returns a list with the Actors of the type that has the attribute "Targets Type".

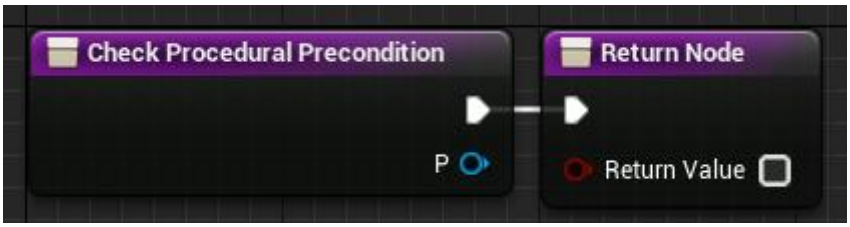
Example:

We use GetTargetsList in CheckProceduralPrecondition (explained later) to compare all the actors in the game of that type, find the one closest to the pawn and set it as target.

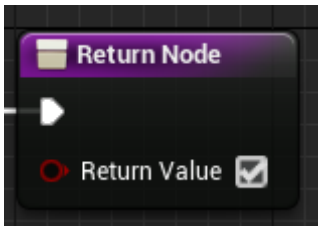


CheckProceduralPrecondition

Customizable function, in case the developer wants the action to have additional conditions to be carried out, it is possible by implementing them in this function, it can also be used to implement preparations prior to the performance of the function, for example deciding the most optimal target.

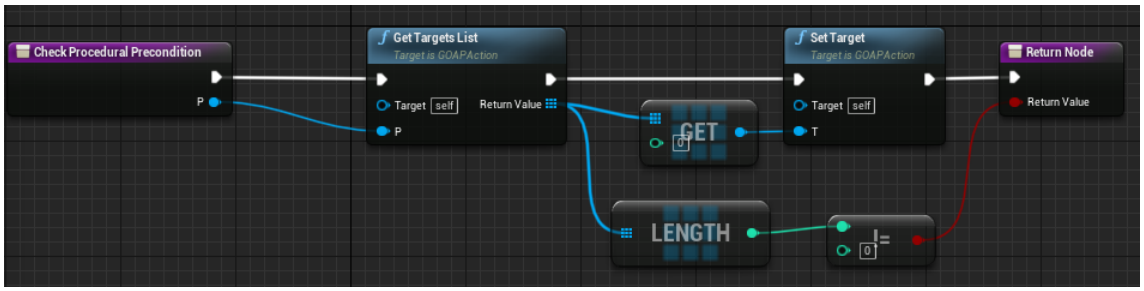


In case you don't want to add additional logic to the precondition of an action, just change the return value to TRUE.



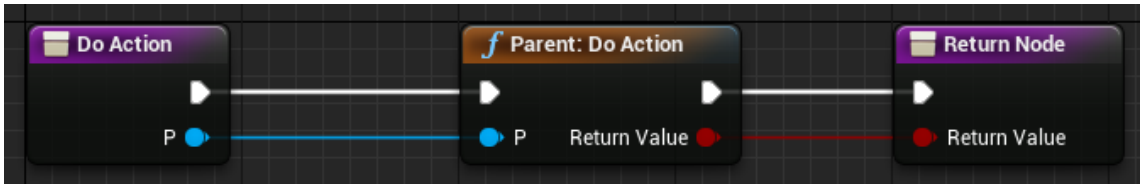
Example:

The precondition added to “Get Coin” action is that exists a coin and at the same time that we perform the check, we assign the coin as the target actor, so that in the doAction we will simply have to implement the logic of going to the target and taking it.



DoAction

Function that contains the procedure and logic that defines the action itself. It does not change the states of the world. It is convenient to remove the Parent node when implementing it.



Example:

"Follow Player" action when executed, removes the previously established target and goes to its position, returns FALSE because it’s never completed unless it’s canceled .