synapse

Documentation

VERSION 1.0

What is synapse ?

Synapse is an artificial intelligence plugin for Unity specialized in the decision-making process. It will help you to design the way your NPC chooses one or another action. To do that, Synapse provides you a visual representation (graph) of each desire and the interest for each one of them so you don't have to write code for this part. For more information, see the documentation section.

The documentation can be found online with updated content. See Contact chapter at the end of the document.

Get started

Get synapse

At first, download the Synapse package from the Unity Asset Store or from the download section of the website.

You have now a folder named Synapse and a subfolder named Synapse-Demo in your project. The second one is a simple demo that helps you to understand how it works. You can remove it from your project if you are already familiar with Synape (and you are ready to go). I suggest not to add or remove other files but the demo folder.

If you are not familiar with Synapse, I will introduce you how it works by recreating the files in the demo folder:

- SynapseAgent.cs
- SynapseDemo.asset
- SynapseDemo.dll

This demo is a simple example of how Synapse helps you in the decision-making process. This simple world contains a NPC, and randomly spawn balls on the ground. Each ball has a value between 1 and 5 points (from dark red to green). The NPC wants to collect those balls to have the higher score as possible but when he moves, he heats and needs to stop to start cooling. You can launch the Demo to see that.

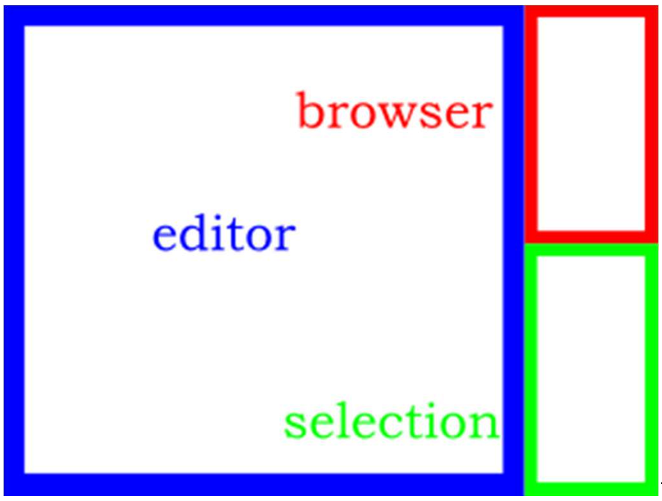
So, let's start! Delete the previous files (or move them outside the Assets folder) but keep the scene file and the World folder intact. You also have to remove the SynapseAgent component in the NPC prefab (in "World/NPC/" folder).

Create the Synapse asset

In the folder where you want the asset to be created, right click and then choose "Create/Synapse library". Three windows will pop.

- **Browser Window** is the window where you can create, rename, delete and order graphs (aka brains) into collections. Collections are just there to help you sort brains.
- **Selection Window** is the window where you can edit the current selection.
- **Editor Window** is the window where you can edit the brain.

I would suggest you to rename the created file and organize the windows like that.



t

1) In the Browser Window, create a collection (using the tool bar of the window) and name it “Demo” (using the Selection Window). Create a brain in this collection and name it “NPC”. When a brain is selected, the Editor Window becomes active to draw a graph.

2) The Editor Window is divided in two spaces. The bottom one is the creation tool bar when all blocks are displayed. The space left is the drawing area. In the right top corner of the drawing area, you can see a box named “Layers”. Layers are important in Synapse, they add a context to a desire of a NPC. For instance, our NPC wants to picks balls that spawn on the ground. This desire of “Pick” is dependent from the ball because the interests to pick one or another ball are not the same (depending on their value, the distance from the NPC…). So we need to create a layer for the balls. You may name it “Balls”, or “Collectibles”.

3) In this layer we will add the desire Pick. For that, go to the “Desire” section in the creation toolbar and drag and drop the empty desire to the right side of the drawing area. Name it “Pick”.

4) An empty Desire block has 2 inputs. The first one is the motivation : the importance of this goal for the character. Drag and Drop a “Constant/Fuzzy” block in the drawing space. Use the mouse to fill the gauge of the constant block to set it around 75% (you may adjust it later). Then connect the output of the constant block to the motivation input of the desire by clicking the input or the output and dragging it to the other one. An orange link should be visible.

There is a color code in Synapse. It helps to understand what kind of data we are manipulating. The orange one is a fuzzy value, the red one is a boolean, the blue one is an integer, the green one is a float, the yellow one is a UnityEngine.Vector3, the black one is a user data (System.Object).

5) The second input of the Desire block is the interest. The interest is dependent of the context. What makes a ball more interesting to pick than another one? There are plenty of reasons but for this demo we will keep only 2 of them. A ball is more interesting than another one if the value is higher and the distance to it is lower. With this sentence, we have made our first step to the artificial intelligence. So we will start by creating a And operator block ("Operators/AND"). Then connect the output of the operator to the interest input of the desire.

6) The And operator only accepts fuzzy values as inputs. We will have to convert "the value is higher" and "the distance to it is lower" to fuzzy values. A ball's value is in the integer domain [1;5]. A way to convert an integer to a fuzzy value is to use a linear interpolation. For that, there is a block that you can find in the "Converters" section. Drag it to the drawing area. This block has 2 outputs. In the Selection Window, there is a Help Box that describes you each output and their relation to each input. A ball is more interesting if the value is higher so we will use the first output: connect it to the first input of the And operator.

7) The interpolation will be between 0 (because even if the minimal value is 1, a ball is interesting) and 5. Create two constant integer blocks with the values 0 and 5 and connect them respectively to the second input and the third input of the interpolation block. The first input is the value we want to interpolate. It is a data that changes depending on the ball. For that, create a integer sensor block ("Sensor/Integer"). The image was blue on the creation tool bar and is now orange. Don't mind, I will explain it later. Sensors are data that Synapse will get from your components. Name the block "Value" and connect it to the first input of the interpolation block.

8) Start the other branch of And operator with an interpolation block (but for float values: the distance is not an integer value) and set the interpolation domain (the ground has a size of 30 by 30 so I would suggest [0;45] to cover the diagonal). This time, a ball is more interesting if the distance is lower. So we will use the second output of the interpolation block. If you have already connected the first one, click on the center of the connection and use the little delete button that appears.

9) Now we have to get the distance but the distance is not a simple value to get. We are going to need another block ("Misc/Distance"). Create it. The block has 2 outputs : the distance between the two inputs and the squared distance. We will use the first or we have to change the interpolation domain. It is up to you. For the inputs, create two Vector sensors ("Sensor/Vector") and name them "Position". Now I explain you what is the difference between the orange and the blue sensor. An orange sensor is a data relative to the current layer (the ball) and a blue sensor is relative to the layer Self (the NPC). This property may be changed in the Selection Window or by right clicking on the image when the sensor is selected. Do that for one of them.

10) Last thing to do is to know which ball has been chosen. Select the Desire block and add an user data input using the Selection Window. A new black circle has appeared on the block. Create a sensor named "Self" and connects it to the new input. This sensor should remain orange because we want to set the ball, not the NPC.

This part may seem long but I have explained in details what to do. With few experience, doing that graph again should take you less than one minute.

The graph is automatically saved after each modification.

Generate the DLL

In the Inspector Window of your asset, there are 2 buttons. Those buttons will generate a dll of the library. Click the debug one. A dll is created in the same folder where the asset is. And… it's done.

Link the DLL to the code

In this section we will see how to connect sensors and desires to the actual code. Create a new C# script named "SynapseAgent".

Making a decision is not something you have to do during each frame. You can do it periodically, or on event. In this demo, a decision can be made only if a new ball appears or when a ball has been picked. But this demo is simple and usually, you will have to update it periodically.

First we need some variables:

- a speed value that can be set by the user to determine at which speed the NPC can move.
- a private field that contains a reference to the CharacterController of the NPC.
- a private field that will contain the interface of the brain (use the type Synapse.Runtime.Brain).
- a private field of type CollectibleValue that will represent the current target of the NPC.

The instantiation of the brain will be done in the Start function with other variables initialization. The brain will be updated each second but you can change this in the script below.

```
// variables
public float m_speed;
private Brain m_synapseBrain;
private CharacterController m_controller;
private CollectibleValue m_target;


// start function : initialize variables and create a coroutine to update the brain
IEnumerator Start()
{
    m_target = null;
    m_controller = GetComponent< CharacterController >();
    m_synapseBrain = new SynapseLibrary_SynapseDemo.Demo.NPC(this);

    while(Application.isPlaying  &&  m_synapseBrain != null)
    {
        AIUpdate();
        yield return new WaitForSeconds(1);
    }
}
```

```
// AI update : update the brain
void AIUpdate()
{
    if(m_synapseBrain.Process() == false)
    {
        m_target = null;
    }
}

// update the movement of the NPC to his target
void Update()
{
    transform.position = new Vector3(transform.position.x, 0.0f,
                                                    transform.position.z);

    if(m_target != null)
    {
        Vector3 velocity = m_target.transform.position - transform.position;
        velocity.y = 0.0f;
        velocity.Normalize();
        Vector3 currentVelocity = velocity * m_speed;

        m_controller.Move(currentVelocity * Time.deltaTime);
    }
}
```

The brain to create is in the namespace SynapseLibrary_ and has the name you have set in the Selection Window.

In the function AIUpdate, we call the method Process on the Brain object to decide which desire is the most interesting. If none is selected, the Process function returns false. The process of the brain will set the variable m_target. Now we will write the function the brain needs to evaluate each desire.

During the creation of the brain, there is a parameter. It references on which object callbacks will be called (self sensors, desires, layers callbacks). You can set another object such as another component but the following functions has to be written in this other component script.

First, we have created a layer earlier. This layer is a context for our desire Pick and represents a ball. To get all balls for the layer we write a function with this signature: object[] GetLayerData() :

```
object[] GetLayerCollectiblesData()
{
    return CollectibleValue.m_instances.ToArray();
}
```

CollectibleValue.m_instances is a list of balls. See the CollectibleValue script for more information about it, but basically, a ball is added when spawned and removed when picked.

Then, we have created a desire named “Pick”. We need to create a function to be called when the desire is chosen. This function must have the following signature : void DesireCallback( ) :

```
void DesirePickCallback(object a_collectible)
{
    m_target = a_collectible as CollectibleValue;
}
```

We have added a user data to our desire so the function has 1 extra parameter of type object.

And finally, we need to create the functions for each sensor. We have 4 different sensors :

- Layer_Ball.Value
- Layer_Self.Position
- Layer_Ball.Position
- Layer_Ball.Self

The second one is the only one that is associated to the layer Self (sensor with blue image), so it will be the only one to have the callback in this script. The signature is the following : void GetSensorData( ) :

```
void GetSensorPositionData(out Vector3 a_position)
{
  a_position = gameObject.transform.position;
}
```

For the last sensor, we used the keyword Self to name it, so we don't have to create a function. Synapse will get the object by itself. For the two others, the function callback must be written in the script CollectibleValue. Make sure the script contains the functions GetSensorPositionData and GetSensorValueData.

Play

You now have all the needed files. If you start the simulation, you will see a NPC moving from ball to ball with no rest (because he has only one desire: pick). From now, you should be able to design the interest for the desire "Cool". This desire does not depend on a specific ball so this desire should be in the layer Self. A simple sentence should help you: cooling is more interesting when the heat is higher. If you want to check what you have done, download the demo files again and check. Don't forget to build the library after each modification made on it.

Runtime classes

namespace Synapse.Runtime

Brain

- **bool Process() :** this method runs through the associated set of graphs to select the most interesting desire at the moment. It returns true if a desire has been selected, false otherwise. If a desire has been selected, the callback named from the desire will be called.
-

namespace SynapseLibrary_<asset_name>.<collection_name>

<brain_name>

- **new (ctor)(object) :** is used to create the brain. The parameter given is the agent object of the brain. It should be a component but can be the game object or anything else you want (but should stay alive until the brain is deleted). Desires, sensors and custom blocks that don't have the static attributes will look for their callback into the instance given.

Contact

Website: http://synapse-plugin.com

Mail : sylvain.minjard@synapse-plugin.com

Thanks to Gladys GUILLET, Unity Technologies and the Unity Community.