State Tree AI

This documentation has the intent to clarify the structure and logic of the project StateTreeAI by Ricardo Avelar that can be found at this github repository.

The project implements six basic features:

1. **Patrol**: The AI patrol a set of predefined waypoints in the scene;
2. **Detection**: The AI detects player presence via line of sight and transit to a chase state;
3. **Chase**: Upon detecting the player, the AI navigate towards him using Unreal's navigation system;
4. **Attack**: When close enough, the AI performs an attack animation using **GAS** and applying an effect on the hitted actor;
5. **Recovery**: If AI encounters navigation problems (i.e. navigation path becomes unavailable) it returns to the navigation so it can back to patrol;
6. **Runtime Interaction (feast)**: During the runtime, the AI detects a corpse around and "eats" it to gain Health.

The visual aspect was not taken into consideration.

Used plugins

The suffix AS means Avelar Studios, which is my one-person company.

CoreAS

CoreAS is a general purpose plugin **created and maintained by me** that implements common logic that all projects may need. Here, it is mostly used for logging/debugging.

NavigationAS

NavigationAS is a robust multithreading plugin that **I am creating** for another project. It is not ready for production yet, but still I decided to use it since it is a perfect fit for the Recovery feature.

Implementation

Patrol

The patrol implementation uses two tasks and three steps. Inside StartPatrol, first we define the character max walk speed to the desired patrol speed and use UPatrolComponent to define the first patrol point (the closest to AI). Then, we wait with a delay to make the movement a bit more natural. Finally, we use a MoveToLocation with UPatrolComponent to get the next patrol location and update the next point.

Detection

Inside ABaseAIController we create and make the perception setup for PerceptionComponent. Also, at ABaseAIController::PerceptionUpdated we set the Target variable, used inside the StateTree to check whether this Target is valid, so we can transit to the Aggressive state.

Note that in some states, such as Patrol, we check this condition to transit to the Aggressive state.

Chase

A simple MoveToActor, but with the difference that we check every tick if we are still on a valid path. If not, we go to recovery.

Attack

Here we call the ABaseAICharacter to activate the attack GameplayAbility and we wait for this ability to end. This ability just plays a montage that calls the UAIAttackNotify, and this notify returns to the ability the tag that allows it to make a trace hitting the actors in front. Note that hitting the player character is not implemented, since the player character has no health, which is trivial to implement.

Recovery

This task just calls ABaseAICharacter::StartRecovery function, that finds the closest navigation point to the AI, and allows NavComponent to do the magic. Just a brief explanation of what NavComponent do under the hood, it creates a Graph in a parallel thread each X seconds, and it uses this graph in other thread to find the shortest path to a point or actor with A*, each Y seconds.

Runtime Interaction (Feast)

Feast uses the new Utility feature of StateTrees to decide if it should search for an InteractiveEffect actor of the type Feast around. Feast is the name of the feature that makes the AI “eat” a corpse on the ground, getting extra health after eating. With a simple implementation, it first finds the corpse around, then the AI walks to it, then plays the eat anim and destroys the corpse.

How to test

On starting the package build you will spawn in a linear designed map, where you just need to walk to the right to see how AI behaves in different scenarios. When going close to any AI, you can try the Chase and Attack features.