THINK INDIA JOURNAL

ISSN: 0971-1260

Vol-22-NO-30 Special Issue-December-2019

Artificial Intelligence in Unreal Engine: A Survey

Roopam[1], ParmohitDev Singh Slathia[2],Manpreet Dhindsa[3]

Department of Computer Science and Engineering

Lovely Professional University, Punjab

roopam.23828@lpu.co.in[1], parmohit7@live.com[2], manpreet.23551@lpu.co.in[3]

Abstract

This paper provides a survey of previous work on the components and working of Artificial Intelligence in Unreal Engine. The material focuses particularly on the elaborating the components used by and to build an Artificial Intelligence Character or" thing" which can be used in the game playing. This approach, I believe, allows game designers/developers to deploy the "enemy" consisting whatever abilities they want it to be deployed with and also easies much of their work in identify rewarding skills that can be explored. This paper covers learning techniques/methods that can be used by any designers or developer ranging from beginner to advanced blueprint scripting. However, the space and time constraints prevent me from demonstrating the detailed introductions of the used learning techniques. Overall, I aimed at reducing the ambiguity of this particular topic that baffled my instincts because of its interesting output behavior.

Keywords: Behavior Tree, Blackboard, Pawn, Navigation Area, Nodes, Context, Generators, Tests,

1. Introduction

Creating an AI is not a big deal when you are given with the best assets by the engine itself. All you have to do is understand that programming language and deploy the solution to each problem you wish to create. But this thinking of mine is conceded with the fact to create an AI which could take the real-time data from game environment and use it to their convenience. Finding your way into a dynamic and complex environment of game (Shooter) is a big Task, especially if you are having very few CPU ticks to spare every frame with an AI actor. How an AI actor will know where to go? What if there are an number of places to fluctuate to, how will can it tell which one is best? Multiple enemies in front, which one it should kill first. It's a similar task to building a perform routine that will give you supply of AI with all the data facts it requires, at low CPU time expenses, while being easy and flexible to use. It requires to filter out the components available in game and filter them according to its spatial awareness while not taking much CPU time and is also intuitive for designers.

ISSN: 0971-1260
Vol-22-NO-30 Special Issue-December-2019

2. Existing System

Prerequisites for this research paper is to have knowledge of Blueprints, working with the pawns, actors, and logic. Creating Artificial Intelligence for characters in the project in Unreal Engine 4 is accomplished through multiple systems working together. From taking different decisions or actions and branching them (Behavior Tree), getting information about the environment (Environment Query System), to deploying queries to extract the sensory information such as sight, sound, or damage information (AI Perception); all these systems played a key role in making of an AI in my project. Let's start with the introduction to the main components of an AI in UNREAL ENGINE 4.

2.1. Behavior Trees

Creating an AI in Unreal Engine can be done in number of different ways. One way is to make the character/pawn do this by YOU giving them the commands. For example, instructing them to "do something" such as play an animation, move to a specific location, react when being hit by something and more. Here the use of Blueprint Visual Scripting is seen. When you want or expect the AI to make its own decisions during run-time, this is where Behavior trees comes as a protector. This asset of Unreal Engine 4 can be used to build an AI for non-player characters in game. Behavior trees are used to execute the branches of logic (I am referring to the real TREE structure here with logic in its nodes and linked to each other). Behavior tree only makes the decision to traverse through these branches or logics. The main asset which serves as a "Brain" for behavior tree is called **Blackboard.**

The Blackboard contains several user defined Keys that holds information used by the behavior tree to make decisions. It could be a Boolean, Vector type or any else depending on the need of the situation. For example: let's take an example of Boolean key type for deciding whether the player is in line of sight or not. If true, AI could attack it. If false, AI would execute another situation (Roaming freely in the Navigation area*). Complex schema of the behavior trees can be designed using these blackboard values.

2.1.1. Creating Behavior Trees

Behavior trees are created in same way as we create functions by connecting a series of nodesandadding them, each having different functionality or specification combined to them. The logic is executed by the behavior tree but the information of the variables used is stored in Blackboard Keys. From left to right Behavior trees executes their logic which means the rightmost gets the priority and is executed first. The numerical numbering indicates the same. The upper right corner of the nodes can be viewed and in placed in the graph. Well, there are various functionalities that can be attached to nodes to control their output such as DECORATORS to determine whether the blackboard key is true or not which further determines whether the rest of branch is to be executed or not. Once the Behavior Tree is ready, it needs to be run during the gameplay. This Behavior tree is

ISSN: 0971-1260
Vol-22-NO-30 Special Issue-December-2019

associated with a Pawn through AI controller. We need to make sure to use "Run Behavior tree" function in AI Controller and a Nav Mesh Bounds Volume to the level.

2.1.2. Advantages of using a Behavior Tree

UE4 Behavior Trees are event-driven to avoid unnecessary work every frame. So, this makes our performance better, since the code does not have to iterate over the tree every tick. This is also helpful in debugging, as, only changes to the location of nodes in the tree or to Blackboard values matter.

Secondly, the use of Decorators in Behavior trees as a Conditionals. They make the behavior tree more intuitive and easier to read. Decorators are used on composite nodes which means leaf nodes are action Tasks, therefore making it easier for developer to see what actual actions are being performed.

On one hand, Standard Behavior trees often use parallel composite node to conceptually perform several tasks at once, instead, UE4 Behavior trees uses special node type called Services (Simple Parallel).This all leads to three main advantages to handle concurrent Behaviors: Clarity (easier to read and understand), Ease of Debugging (Graphs with fewer simultaneous execution paths are easier to debug), Easier Optimization (Graphs with event-driven property are easier to optimize.)

2.1.3. Behavior Tree Node Types

All tasks, logic flow control, data updates like services are performed by the Behavior Tree Nodes. First of first, Root node serves as the starting point in a Behavior Tree. Only supported for one connection and no Decorator Nodes or Service Nodes. On its own, it holds nothing but for all other nodes it acts as a base. Four types of Behavior Tree nodes are:

- Composite Nodes: Fundamental basics of a node or branch and the base rules for how the branch is executed is defined by this node.
- Task Nodes: All the leaf nodes of the Behavior Tree that have the tendency to do some actionable things.
- Decorator Nodes: These are also known as conditionals. They stick to other nodes to make decisions on the execution of that node.

Service Nodes: Attached to composite nodes they will execute periodically, mostly used to make checks and to update the Blackboard.

2.2. Environment Query System

This is the real deal when it comes to using AI in UE4. The Environment Query System (EQS in short) is used for collecting data from the environment. Once the data has been collected, system decides "what to ask" and "How to ask" using TESTS and returning the best Item that fits the question. Some Use case can be: On NAVMESH, what positions

ISSN: 0971-1260
Vol-22-NO-30 Special Issue-December-2019

are closer to the enemy so that enemy remains visible from there or prefer those which are hidden.

2.2.1. ETQ

Concept of EQS came from ETQ (Environment Tactical Query) which started in Unreal Engine 3. Originally written by MieszkoZielinsky, ETQ was one of the two systems that were designed and implemented in the early development of *Bulletstorm.* The main goals for the development of the system were: code reusability, performance, let non-programmers do the job. Simple questions were supposed to be asked to data creator: What to generate? ; Who's asking? Where to look? Which items are good enough?; Which items are better?

Query which is to be asked to the ETQ system has three main components and ETQ system is a data-driven answer or result, and most its power lies in data and facts representation.

- Query Template Id: Which one of the user-created questions are we asking? Each question according to its specific enquirer is assigned their unique id and that is what is used.
- Context Object: The game entity that is asking the question.
- Items list: All items list found are returned as a list.

The core ETQ algorithm can be read in the following figure.

Listing 33.2. The core ETQ query algorithm.

```
foreach Option in QueryTemplate.Options:
    Query.Items = (generate items with QueryTemplate.Generator
            using contextual data from Query);
    if Query.Items is empty:
        continue to next option;
    foreach Test in Option.Tests:
        Reference = (find world object Test refers to);
        if Reference not empty or not required by Test:
            //explained in Section 33.7 under "Fail Quickly"
            if Test has a fixed result:
                apply result to all Query.Item elements;
            else:
                perform Test on all Query.Item elements;
            if Test.bCondition is true:
                filter out Query.Item elements that failed Test;
            if Test.bWeight is true:
                foreach Item in Query.Item:
                    calculate weights from every test result
    if Query.Items not empty:
        foreach Item in Query.Items:
            sum up all weights calculated by weighting tests;
        sort Query.Items descending with computed weight;
        return success;
return failure;
```

ISSN: 0971-1260
Vol-22-NO-30 Special Issue-December-2019

Below is the table showing how different test cases can be used as both conditions and weights.

Test	Condition	Weight
Visibility	Is (not) visible	Prefer (not) visible
Distance	More/less/equal to X	Prefer closer/further away
Configurable dot	More/less/equal to X	Prefer more/less
Within action area	Is (not) in action area	Prefer (not) in action area
Reachable	Is (not) reachable with navigation	Prefer (not) reachable
Distance to wall	More/less/equal to X	Prefer more/less
Current item	Is (not) current item	Prefer (not) current item

Although the explained system was verysimple in its own design, it has proved to be very powerful and strong. Without eating a lot of CPU time, a lot of environment querying power was gained. But there were lot of improvements that were needed. Like multithreaded implementation, Multigenerators, using AI's current cover point to grade it first and then accept the very first item that passes all the query filters and has a higher score, merging test problem aroused meaning that certain tests tend to show up as group.

EQS came as an experimental feature in Unreal Engine 4 but is still convenient to use. EQS can be created in Content Browser. We first create the Generator node to produce Items and then we select the desired tests to run on those Items and the Context in which to run them. All this is done to lessen the items returned and need of the processes to score them. There are 3 main Node types in EQS:

- Generator: Produces location of actors, referred to as Items, that will actually be tested and weighted.
- Contexts: Frame of reference for the various Tests and Generators.
- Tests: Used to filter out the Items.

2.3. AI Perception System

This tool, provided by UE4, in addition to the utilities provided by Behavior Trees an EQS (Environment Query System) provides sensory data for an AI. It is a way in which a pawn receives data from the environment such as where noises are coming from, if it his getting damage from something, or if it sees something. In other words, AI Perception System acts as a stimuli listener and gathers registered Stimuli Sources.

ISSN: 0971-1260
Vol-22-NO-30 Special Issue-December-2019

"AI Perception Component" can be added to pawn's blueprint using the Components window. Once it has been added, we can add the type of senses and how they can be perceived. Various senses available are enlisted below.

- AI Damage: Upon receiving any type of damage, AI reacts to the stimuli.
- AI Hearing: Sounds generated by a Report Noise Event can be detected by AI Hearing sense. For Example: When an object hits the ground or something, it creates a sound which is perceived by AI Hearing sense.
- AI Prediction: Function of this is to ask the perception system to supply Requestor with Predicted Actor's predicted location in Prediction Time seconds.
- AI Sight: Using this, AI character can actually "see" things in the level. Whenever the actor is within the radius of AI, the perception system detects it and signals an update.

- AI Team: This helps the AI to distinguish the enemy and ally players of its kind.
- AI Touch: Whenever the AI is grasped/touched/bumped onto by any other actor in the game, this sense comes into play and detects it and helps AI to respond with different logic.

2.3.1. Perception events.

This component of AI perception System is fired up once the AI perception system receives an update or when the AI perception system is activated or deactivated. Four components associated with it are:

- On Perception Updated: All the actors that are causing the update are returned in the for of an array using this function.
- On Target Perception Updated: Perception system receives the update and return the Actor that signaled the update along with the AI stimulus struct.
- On component Activated: Used when the AI perception component is activated.
- On Component Deactivated: Used when the AI Perception component is deactivated.

3. Conclusion

A game is soul-less piece of software until or unless one feels the attracting reason to play it again and again. Unreal Engine not only provide you with an interactive platform to develop a game but it also gives you the opportunity to explore the various tools and assets it supports. While striving to make a full-fledged game not only I realized that developing a game not a piece of cake but also about the different processes involved in doing so. Having a simple AI design is enough for the Developer's satisfaction but in order to quench the thirst of user's thirst, you have to push your game further to its limits.

ISSN: 0971-1260
Vol-22-NO-30 Special Issue-December-2019

Even though the system uses up the resources to fulfill the AI needs but they are worth it and nothing is more important than an enjoyable experience of user. Allowing the EQS to take over the interrogation part, AI can improve its priorities. Also, the debugging tools further enhances this asset. This element of "Surprising" has always captured my attention.

4. To be explored

Data is like a fuel of artificial intelligence. Every piece of data generated in the game is a potential subject of research. How a player interacts with the enemy or other players or with the objects present can be analyzed with smart algorithms and can be used to make the game for fun and entertaining which in turn helps to grow the user base. Game with huge data generation needs to be of cloud-based nature. Although, this approach of using data analyzation has already been deployed in various games developed by Unreal such as "Fortnite" from "Epic Games" of which much of its infrastructure is built on Amazon's AWS cloud service. Various tools and services related to data analytics like Amazon S3, the Apache Spark cluster computing framework and the Tableau visualization software.

5. References

[1] Lee, Myoun-Jae. "A Proposal on Game Engine Behavior Tree." *Journal of Digital Convergence* 14.8 (2016): 415-421.

[2] Assadi, Ramin. "Behavior Trees using Unreal Engine 4."

[3] Andrew Woo, (2012), Behavior Trees and Level Scripting in LEAGUE OF LEGENDS, http://aigamedev.com/premium/interview/league-of-legends/

[4] Unreal Engine 4, (2014), How Unreal Engine 4 Behavior Trees Differ, https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/HowUE4BehaviorTreesDiffer/index.html.

[5] Lewis, Michael, and Jeffrey Jacobson. "Game engines." *Communications of the ACM* 45.1 (2002): 27.

[6] Gruenwoldt, Leif, Stephen Danton, and Michael Katchabaw. "Creating reactive non player character artificial intelligence in modern video games." *Proceedings of the 2005 GameOn North America Conference*. 2005.

[7] Fritsch, Dieter, and Martin Kada. "Visualisation using game engines." *Archiwum ISPRS* 35 (2004): B5.

ISSN: 0971-1260
Vol-22-NO-30 Special Issue-December-2019

[8] Waltham, Michael, and DeshenMoodley. "An analysis of artificial intelligence techniques in multiplayer online battle arena game environments." *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*. ACM, 2016.

[9] Petridis, Panagiotis, et al. "Game engines selection framework for high-fidelity serious applications." *International Journal of Interactive Worlds* (2012): Article-ID.

[10] Herwig, Adrian, and Philip Paar. "Game engines: tools for landscape visualization and planning." *Trends in GIS and virtualization in environmental planning and design* 161 (2002):