Behavior Trees using Unreal Engine 4

http://youtu.be/GGlPuxkuE88

Ramin Assadi

Abstract

This project will describe how to implement an AI with Behavior Trees in Unreal Engine 4. It will be shown how to build an AI with all of its components. The purpose about each component will be explained. This will be implemented in an ongoing development game called Warlock Arena.

1 Background

To understand the AI and what the behavior of it should be, a brief explanation of the game is necessary. The name of the game is Warlock Arena and it is based on a mod called Warlock from the game Warcraft 3. It is a multiplayer game were the player controls a warlock and the objective is to kill the other warlocks that is possessed by other players. They face each other in a arena were the ground gets smaller and smaller. If the player steps outside of the ground they will quickly lose health. The objective is to shoot spells at each other and push players outside the ground. The last remaining player will get the most points. This continues for a couple of rounds and at the end the player with the most points will win. The purpose of this project is to implement an AI so the game can run in offline mode.

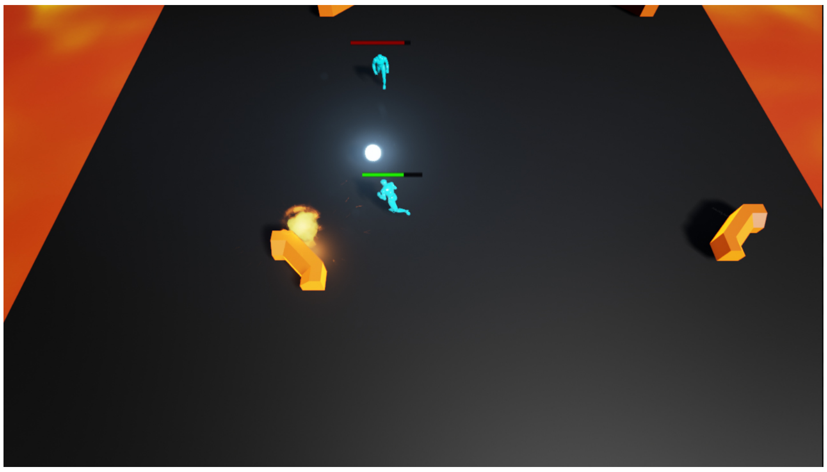


Figure 1: Screenshot from the game Warlock Arena

2 Theory

When constructing a game there is three Hierarchical logics that can be used that is suited for games. These are Scripting, HFSM (Hierarchical Finite State Machines) and Planners HTN(Hierarchical Task Network). Each of these hierarchical logics has its own advantages and disadvantages. When it comes to scripting the advantages is that any computation is possible. But the downside of scripting is when looking inside of the script and figuring out what the scirpt is doing and what it will do. For example planning ahead is not an easy thing to do with scripting. And for designers the scripts are not accessible. A solution to scripting would be to use HFSM. These are great for designers, the only thing they have to do is to take a bunch of states and edit transitions, and plug them together. That gives them all the control they need at the lowest possible level. The downside of HFSM is that it takes a long time to implement all the transitions and states. One possible solution to HFSM is planning. Planning essentially uses a form of search to plug together these modular behaviors to reach any kind of objective within a game. And that provides a certain level of automation. The downside is that planners are more difficult to implement than scripting. The problem here is that an elegant solution in theory can result in a brute force solution in practice. So the problem is that all these hierarchical logics have there advantages and disadvantages. What would be the best suitable solution for a game would be a combination of these logics to fix each others disadvantages. This is were Behavior Trees comes in. This does not mean that behavior trees do it better then the other logics. This is just a solution that makes it easier for game developers. Behavior Trees makes it very easy to make decisions over time, follow out those decisions and making sure the plans executes correctly. This is why Behavior Trees are good for games.

3 Method

The method used in this project is Behavior Trees. There are three different nodes in a Behavior Tree. Composite, Decorator and Leaf. The composite can consist of different children. There are two different composite nodes. Selector and Sequence. The sequence node tells each child to run one at the time until all children succeeds. The selector node runs each child until one returns success then the selector stops. A decorator node, like a composite node, can have a child node. Unlike a composite node, they can specifically only have a single child. Their function is either to transform the result they receive from their child node's status, to terminate the child, or repeat processing of the child, depending on the type of decorator node. The leaf node is the lowest node and therefore has no children. This node will carry out the task that the AI is going to do.
When it comes to Behavior Trees in Unreal Engine 4 there are couple of components that is necessary to have for the AI to be complete. In Unreal Engine 4 [4] there are four different components. Conditions[1], Loops[2], Tasks and Blackboard. The AI needs to store some values that the user sets. In other words it needs a memory. If the AI should be smart and make decisions depending on

[1]In Unreal Engine 4 they are called Decorators
[2]In Unreal Engine 4 they are called Services

what is going on in the game, there needs to be a memory for the AI to store information in. The blackboard is the memory. In the blackboard whatever value can be stored. But the important part is to not store to much information in the blackboard since it increases the time to save and retrieve the data. The tasks in Unreal Engine 4 is the leaf node. The conditions and loops are connected. The conditions are different queries that can be put on the composite and leaf nodes. It simply asks the blackboard for information to see if the value is true or not. It is in other words a if statement. The loops look at the world and change the blackboard depending on what happens. For example a loop can look if the AI has an cooldown on one of his attacks and save the result in the blackboard. Then the condition will ask the blackboard if the attack is on cooldown and if it returns true or not depending on what the condition is, it will proceed or stop.
The one thing that makes Behavior Trees extremely suitable with Unreal Engine 4 is the debugging system. With the debugging system in Unreal Engine 4 it is possible to pause the behavior tree at any point. When the behavior tree is paused the possiblity to jump back and forward in time is possible. While doing the debugging it is also possible to see what the value of the blackboard is at that specific time. This makes it easy to see when and why things happen or not. In other words it is possible to see how the behavior tree behaves at a specific time to pinpoint problems and bugs in the AI.

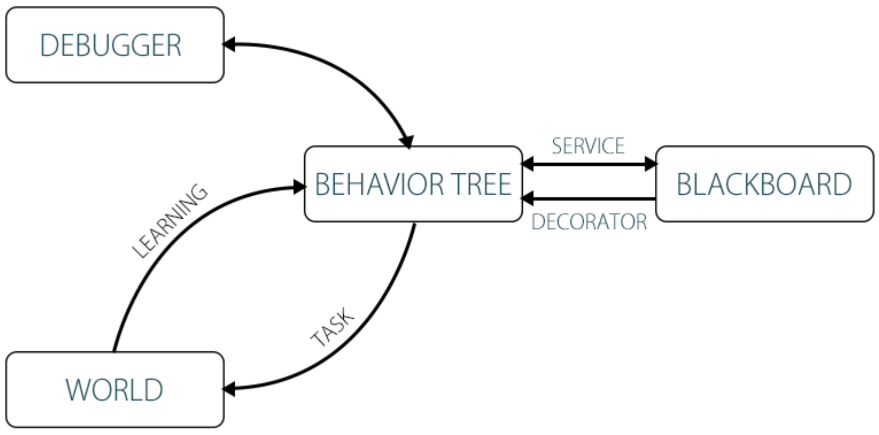


Figure 2: System architecture of the Behavior Tree in Unreal Engine 4

In Fig.2 we can see the system architecture for behavior trees in Unreal Engine 4. What the world represents in the image is all of the information from the game. The behavior tree can learn anything from the world and use it to modify the behavior. We can see that the service can change and recive information from the blackboard while the decorator only can recive information. When it comes to the debugger it can get and send information to the behavior tree.

And when all the interactive parts are done it will end with the behavior tree sending a task back to the world. This task is what makes the AI interact with the world. This process continuous until the AI is removed.
The AI in Warlock Arena is programmed to shoot the closest target whenever his spells are not on cooldown. If the target is to close it will move away appropriately and if the target is too far away it will move closer to the target. The services looks if the spell is on cooldown, which target is the closests and if the target is far away or close. The decorators asks the blackboard what spell is on cooldown and if it should walk away or go closer. The tasks does all the actions for the AI. In this case turns the AI, shoots the spells and moves the AI to the right position.

4 Results

This is the Behavior Tree for Warlock Arena in Unreal Engine 4:

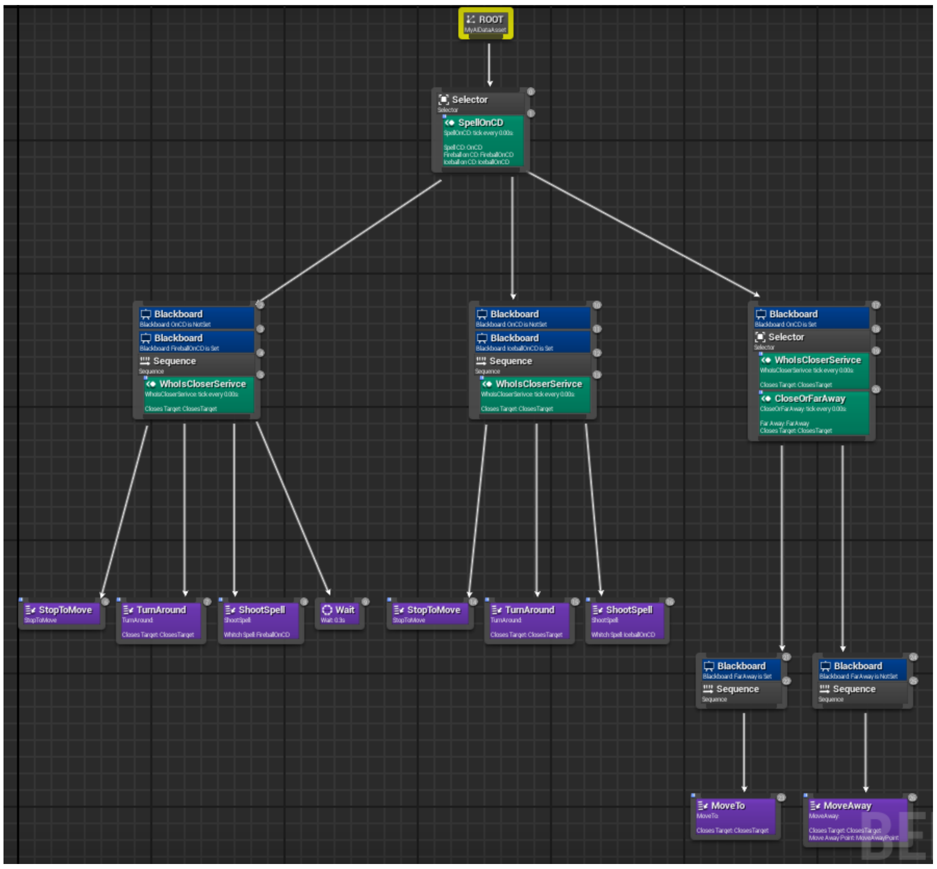
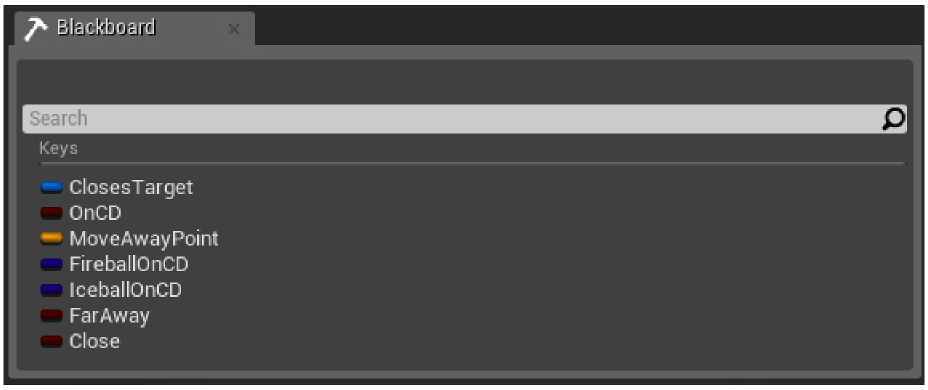


Figure 3: Warlock Arena - Behavior Tree in Unreal Engine 4

In this Behavior Tree in Fig.3 the green nodes are the services, the blue nodes are the decorators, the purple nodes are the tasks and the grey nodes are the composite nodes.


Blackboard

Search

Keys

- ClosesTarget
- OnCD
- MoveAwayPoint
- FireballOnCD
- IceballOnCD
- FarAway
- Close

Figure 4: Warlock Arena - The Blackboard

In Fig.4 we can see the AI's blackboard. This is what is stored in the AI's brain. It is all from bool values, vectors and different object classes. This information gets updated and used when the AI is making decisions.

For a video demo of how the AI behaves in the game, please visit this link:
Warlock Arena - Behavior Tree using Unreal Engine 4

In the video the character with the green health bar over his head is the player. The character with the red health bar is the AI.

5 Discussion

5.1 Result

The result achieved for the AI in Warlock Arena is good. The AI has a brain and can navigate the map depending on how the player moves. It can also predict how the player is moving and shoot accordingly. The result is good but not the best. To make the AI better there needs be more tasks and behaviors implemented. This is not hard to do, but it is time demanding. The structure of the behavior tree and its functionallity is done and therefore implementing tasks and behaviors can anyone do and import it in the behavior tree. The more complicated AI that is desired the more complicated tasks and behaviors is needed.

5.2 Method

The method Behavior Trees brings alot to the table. What makes the method exceptionally good is the ability to easy work with designers that does not have any knowledge about programming. For example the tasks and behaviors are programmed by the programmer and saved in nodes. When a designer wants to change the behavior tree they can use the tasks and behaviors how many times

and how they want. This makes it easy for a programmer to work with a designer when implementing an AI. If the designer wants the AI to be able to shoot the programmer will simply create a task. This task can now be reused whenever and wherever the designer wants to use it. This is what makes Behavior Trees such an easy method to use.
Another aspect of why Behavior Trees are good is the debugging of the tree. For example if there is a problem in a big system it is nearly impossible to pinpoint the problem. By using something called breakpoints it can stop the AI at the specified node and then it is possible to check all the values in the blackboard. It is also possible to jump back and forward in time to see where the problem occurs. This makes it easy to find problems in the Behavior Tree.
There are also some disadvantages with using Behavior Tree. When it comes to bigger systems it is really hard to navigate the Behavior Tree if it contains hundreds or even thousands of nodes. Then tool support is needed to be able to navigate.

6 Conclusion

By examining the different methods we can conclude that Behavior Tree is the right method for this game. The result indicates that our methods and calculations are correct. Although the AI is not complicated, the expansion of the Behavior Tree is an easy task but time demanding. When working in groups and programmers need to communicate with designers, Behavior Trees is a compatiable method to use.

References

[1] *Alex J. Champandard*, (2008), Behavior Trees for Next-Gen Game AI, http://aigamedev.com/insider/presentation/behavior-trees/

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

[3] *Emil Johansen*, (2013), If You Can Build a Behavior Tree, You Can Dodge a Ball, http://aigamedev.com/premium/interview/behave-dodgeball/

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