What are the advantages and disadvantages to using behavior trees in simple NPC programming for digital games?

COMP160 - Software Engineering Essay

1506919

March 28, 2017

This paper discusses the advantages and disadvantages of using behavior trees for managing simple NPC AI behavior. First there will be an introduction of behavior trees, what the main structural features are and what they do, also how a game engine would run the behavior tree. The paper then names the advantages and disadvantages of using behavior trees for AI character behavior compared to other AI NPC programming methods

1 Introduction

One of the deciding factors on how well a game is rated is the behaviour of the NPC AI within the game, their movement and senses have to be realistic[1], e.g. if the NPC enemy is supposed to detect the player with sight, it would be wrong for the player to be detected through a wall. Therefore programming an NPC is one of the most important parts of producing a game[2] and should

be simplified as much as possible, unfortunately there are many different ways to programme an AI controlled character, of which behaviour trees is one of the most recent methods[3], this paper will look into the advantages and disadvantages of simple NPC programming using behaviour trees. The conclusion will discuss whether the advantages of behaviour trees outweigh the disadvantages and the future of NPC AI programming.

2 What are behaviour trees?

Rahul Dey and Chris Child state that behaviour trees (BT) were created as a more intuitive revision of a finite state machine[1], and are actually a form of hierarchical state machine. A brief description of a final state machine is a number of states that require an input to transition to another state which may result in an output or action[2]. BTs improve on finite state machines by using hierarchical system which gives better control over AI behaviour, the basic structure consists of 4 main node types[4]; leaves, composite, decorator and root. When the BT is called it runs through the nodes from left to right, therefore the higher priority nodes are placed on the left which means they will be checked to see if they are able to run first, if the higher priorities cannot be run the BT will start again but going to the next node to the right of the previous explored branch.

2.1 Root

A BT starts with a root, it has no parent which means there is nothing before the root, there can only be one root per tree and the root can have one or more children, which means the nodes that the root feeds into. Nodes are attached to each other and the root via branches this is how the BT knows which node is next to be explored. (Depending on which game engine you

build a BT in could change how nodes are placed and what they do.)

2.2 Composite

Composite nodes can have one or more children and the main responsibility is to depict how the branch will be run. There are three types of composite nodes.

Selector (sometimes called Priority): Returns success or failure depending on the children's states, it will succeed if one or more child has succeeded and fail if all children have failed[5].

Sequence: If one of its children fails the sequence node returns as failed.

parallel: Instead of checking its children on after another parallel runs all children at the same time.

2.3 Decorator

Decorator nodes only have one child, and the main purpose of a decorator is to manipulate the child's result, this could mean if the child succeeds the decorator says that it failed or repeat the child a certain number of times.

2.4 Leaf

Leaves are the actual actions or behaviour that the NPC exhibits in game they have no child which means after their action is completed the BT is started at the root again[6].

3 Advantages of behavior trees

BTs are considered to have better modularity compared to other NPC programming methods, all the nodes can be synchronised inside the same BT and each node

is responsible for one aspect of the behaviour function[7]. The advantage of the BT structure is that the design created uses functional requirements, this very significantly reduces the complexity and time of the layout process and which means the time spent making any changes can be greatly reduced when using BTs for NPC AI behaviour[8], other programming techniques e.g. node based scripting language can take many hours to organise each node to create the best readable layout, depending on how many nodes are used in the NPC behaviour. Large BT can be split into smaller trees which can be easier to understand and expand upon[6]. BTs have the ability to run nodes at the same time as others, this is something the finite state machine programming is not capable of but by can be possible if a BT is used along side the finite state machine using a parallel node[9]. BTs have the benefit of being available to use in many game engines, which means with a reasonable amount of knowledge of BTs an individual programming AI behaviour would be able to use different game engines with not a great amount of difficulty.

4 Behavior tree disadvantages

- A large amount of nodes arranged in big groups within a single BT quickly becomes unmanageable and when split into smaller manageable BT it is not immediately obvious how to get the two or more trees to communicate[8] and can often be quite difficult[10].
- Most nodes within the BT have the ability to have two parents which can make the structure much harder to read[6]
- Bugs can be hard to find/ fix as BT get larger and more complicated[3]
- BTs can become quite detailed and much larger than can be used in a game

- In newer games, studios are starting to want more detailed AI that makes more realistic decisions to limited environmental information which BTs just can't deliver as the method gets older[11].

5 Conclusion

Utility AI is starting to take over from behaviour trees [11]
[12] [13] [14]

References

[1] R. Dey and C. Child, "Ql-bt: Enhancing behaviour tree design and implementation with q-learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on.* IEEE, 2013, pp. 1–8.

[2] M. Buckland, *Programming game AI by example.* Jones and Bartlett Learning, 2005.

[3] D. Isla. (2005) Managing complexity in the halo 2 ai system.

[4] C. Simpson. (2014) Behaviour trees for ai: How they work. Accessed: 12/03/2017. [Online]. Available: http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

[5] Unreal engine - behavior tree quick start guide. Accessed: 25/02/2017. [Online]. Available: https:/docs.unrealengine.com/latest/INT/Engine/AI/BehaviourTrees/QuickStart/

[6] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 5420–5427.

[7] M. Colledanchise, R. Parasuraman, and P. gren, "Learning of behavior trees for autonomous agents," *arXiv preprint arXiv:1504.05811*, 2015.

[8] R. G. Dromey, "From requirements to design: Formalizing the key steps," in *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on.* IEEE, 2003, pp. 2–11.

[9] What is the difference between fsm and behavior trees? Accessed: 27/03/2017. [Online]. Available: https://www.scirra.com/forum/what-is-the-difference-between-fsm-and-behavior-trees_t124518

[10] P. Mawhorter. Behavior trees and reactive planning. Accessed: 28/03/2017. [Online]. Available: https://www.cs.hmc.edu/~pmawhorter/research/slides/behavior_trees-Mawhorter-2010.pdf

[11] J. Rasmussen. Are behavior trees a thing of the past? Accessed: 28/03/2017. [Online]. Available: http://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php

[12] A. Einhorn. Introduction to ai - part 1: Behavior tree basics. Accessed: 25/02/2017. [Online]. Available: https://www.youtube.com/watch?v=2K6nX4A5n8w

[13] M. Nicolau, D. Perez-Liebana, M. O'Neill, and A. Brabazon, "Evolutionary behavior tree approaches for navigating platform games," *IEEE Transactions on Computational Intelligence and AI in Games*, 2016.

[14] A. Nareyek, "Ai in computer games," *Queue*, vol. 1, no. 10, p. 58, 2004.