**DAC619**

**AI Assessment**

**Kevin Buchholz**

Table of Contents

[1 Behaviour Tree 3](#_Toc26052722)

[1.1 Justification 3](#_Toc26052723)

[1.2 Advantages 3](#_Toc26052724)

[1.3 Disadvantages 3](#_Toc26052725)

[2 Comparison 3](#_Toc26052726)

[2.1 Decision Tree 3](#_Toc26052727)

[2.2 Utility AI 3](#_Toc26052728)

[3 Analysis and Design 3](#_Toc26052729)

[4 Conclusion 3](#_Toc26052730)

[4.1 Strenghts of Behaviour Tree 3](#_Toc26052731)

[4.2 Weaknesses of Behaviour Tree 3](#_Toc26052732)

[4.3 Improvements to the AI 3](#_Toc26052733)

[5 Appendices 3](#_Toc26052734)

[6 References 3](#_Toc26052735)

# Behaviour Tree

## Justification

For this AI creation task, we have many different techniques on hand. We can use these following methods:

* Switch Case
* State Machine
* Decision Tree
* Behaviour Tree
* Utility AI

I choose the Behaviour Tree as my approach for the AI since it is combining many basic approaches we use in the other techniques and it is the most common approach for AI creation in today’s game industry. Likewise, it will be very useful for usage in future applications. The Switch Case and State Machine are in my opinion far too easy to implement and not the greatest option to pick here. I want to have a little challenge and learn new things while working on this assessment. The Decision Tree is a more simplified version of the Behaviour Tree without the versatile options it has to offer, so we do not choose that either. The Utility AI is maybe a more suitable option for the future of video games but currently it is too sophisticated for this project in my opinion. A single Behaviour Tree will suffice for this project.

## Advantages

Behaviour Trees offer highly reasonable and flexible options for creating more complex AI patterns. We have two core modules known as Sequence and Selector. Based on this we can build easily every pattern we want to implement for the AI. Of course, there are more optional components like the Parallel module, but we will look at this later if we need to implement this. If the Tree is clever designed, the AI can decide between different branches easily and always achieve the expected and best result. Additionally, we are able to create a visually impressive and very understandable tree by using a graphical user interface for it. Then even non-programmers can use this method for creating an AI architecture.

## Disadvantages

On the other hand, a single Behaviour Tree can get really complicated over time when building a more and more complicated AI structure. At that point it can be very difficult to debug the AI in its entirety without breaking down each branch of the tree. Furthermore, even a single AI agent can be a huge performance impact on our system, if the tree is not performance friendly programmed.

# Comparison

## Decision Tree

A Decision Tree can offer many viable options in creating an AI, but its core mechanism is limited to binary operations. This means the tree can create branches by evaluating the data as “Yes” or “No” only. We have just two options for each branch here. So each node just relies on a false or true result. In addition, we must be careful by not creating too many branches without planning it first. Even a single input change can influence the whole tree. It has to be balanced on each side of the tree to make it more stable. Instead for the Behaviour Tree we can create as many branches as we want without any particular order. We have to look how we utilise the Composite Nodes to produce an understandable and convenient process. However, a Decision Tree is highly more understandable at first glance because it does not have those complex patterns and follows an easier creation process. Therefore, it is easier to debug than the Behaviour Tree because we can determine the node, we are currently evaluating more easily than by the complex design patterns of a Behaviour Tree.

## Utility AI

The Utility AI is a very modern approach for creating complex and coherent structures. It does not follow a tree creation process. Instead the AI relies on determined desires. We set different goals for the AI that it must reach by increasing or reducing these desires. Thus, this approach can easily blend between different actions if the AI needs it in the current situation. It can look at more conditions at the same time than the Behaviour Tree. The order in which the inputs get evaluated does not matter. The Utility AI can perform any action by just looking on the given score. Hence, the AI can make decisions without any predefined routes it has to follow. The Behaviour Tree pursues the exact opposite procedure. There you have to plan every step the AI can possibly make. Referring to the Utility AI it can choose actions the player does not have foreseen as well.

# Analysis and Design

Our AI has to play the famous game mode “Capture the flag” against its opponents. Basically, there are three main paths the AI must follow:

* First, try to steal the flag from the enemy base
* Second, bring the stolen flag to his own base
* Third, if own flag gets stolen, chase down / attack the enemy and recover flag

We have to build these three paths but have to consider some possible tactics and different approaches the AI can evaluate while fighting the enemy on its way. For this we have to carefully build reliable branches within the Behaviour Tree where the AI can easily switch back and forth.

# Conclusion

## Strengths of Behaviour Tree

## Weaknesses of Behaviour Tree

## Improvements to the AI

# Appendices

Behaviour Tree Chart

Pseudo Code

Test Plan

# References

**GAMASUTRA, 2016. [viewed 30 November 2019]. Available from:**<https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php>

**GAMASUTRA, 2014. [viewed 30 November 2019]. Available from:**<https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php>