Understanding and Evaluating Behaviour Trees

Jonathan Tremblay
Jonathan.Tremblay2@mail.mcgill.ca

Abstract

Behaviour trees is a formalism mostly used by the game industry to model the behaviour of non-player characters. Since it is used by independent communities, it lacks consistency in the definition of the model and almost no metrics are provided.

This report addresses these issues by presenting a well defined formalism for the behaviour trees and environment and non-environment oriented metrics. Moreover, the metrics were tested inside a game simulation.

Keywords: Artificial Intelligent (AI), Decision Making and Behaviour Trees

1. Introduction

Behaviour Trees (BT) is a formalism used by game and robot developers, and modeller to model different Artificial Intelligent (AI) comportment. Although, it is mainly used by game developers for its ease to use and understand by non-programmers. BT have a lot of in common with Hierarchical State Machines but, instead of state, the main building block of a behaviour tree is a *task* [5]. A task can be something as simple as looking up a variable or executing a particular action. The organisation of tasks into a sub-tree will results into a complex action such as opening a door or set explosives. Moreover composing a tree of sub-trees will result into a complex behaviour such as choosing to chase a deer in order to eat its meat. Even though BT are mainly used by the game industry, the formalism comes from the Robotics world [2].

Since the formalism is mostly used in the game industry [3], it lacks of good documentation and consistency. For example, the meaning of task differs from authors [1, 4]. Moreover, it lacks tools to evaluate the profile of BT [6]. This report tends to address these issues by presenting BT as

a defined formalism and by giving metrics that are environment related to profile BT. In order to test the metrics a simple game was implemented and tested.

This report is organized as follow, the second section introduces BT as a defined formalism. The third section presents our implementation. The fourth section talks about our experiment and the metrics used. In the last section, there is a conclusion and an opening on further works.

2. Behaviour Trees

For the last 10 years, the formalism used by game developers to build different AI comportment is BT. This success comes from the simplicity to understand, use and develop BT by non programmers [5]. A BT is formed by internal and external nodes. An external node or task, could represent anything from walking to a specific point, see any enemies, shoot, etc. Tasks are composed into sub-tree to represents more complex actions. Again these actions can be reorganized into higher level behaviour. This low-level to high-level composition is powerful, when building higher level behaviour, it removes the awareness of implementing low-level task as they are independent.

It is important to note that BT lives inside discrete time. Moreover, in the case of game, this time based is so fast that for any outsider observer this seems to be continuous, for example 60 frames per second.

Also, it is important to note that there is a notion of a blackboard (state) influenced by BT. For the following, we are going to assume that this blackboard is invisible.

2.1. Types of Task

Every tasks have some basic structure in common, they all have some function to run and they return the status of their success or failure, boolean structure could be used.

$$\text{task} =< f, out > \tag{1}$$

In equation 1, f references to the task function and *out* is a boolean variable based on the success or failure of f. A behaviour tree consist of different kind of tasks, inside this report we will consider: Condition, Actions and Composite.

Condition. A condition task test some property of the environment. For example, do I see an enemy, do I have more than 50 health, can I walk there, etc. As a rule of the thumb, the condition should test one property so it can be reused. The node has to return the status (true or false) of the property checked.

Action. An action alter the state of the environment. There can be actions for animation, movement, changing the internal state, shooting, etc. Most of the time, an action task will return success, if there is a chance not, it is important to check with a condition before executing the action.

$$[\![\bigcirc]\!] = out(f) = \begin{cases} \text{True} \\ \text{False} \end{cases} \tag{2}$$

Both condition and action tasks sit on the leaf node of BT. They are denoted with the name of the task inside a circle. In equation 2, the meaning of a task is denoted as True or False based on the output of f.

Composite. Most of the branches of BT are composite tasks and their behaviour is based on their tasks children.

$$\text{Composite Task} =< f(c), c, out > \tag{3}$$

In equation 3, c refers to the children of that task, it is define by the modeller and respects a certain organisation, such as first in first out. The boolean variable *out* is based on the success and failure of $f(c)$ and $f(c)$ takes into consideration the children tasks. For this project, we consider the following special composite tasks: Selector, Sequencer and Decorator.

Selector. A selector task will return immediately with a success (true) status code when one of its children runs successfully. It is denoted with an interrogation point inside a circle.

$$[\![?]\!] = out(f(c)) = \begin{cases} \text{False} & \text{if } i = \text{False} \quad \forall i \in c \\ \text{True} & \text{otherwise} \end{cases} \tag{4}$$

The meaning of a selector task is expressed by equation 4, if one of the children is true, it will return true. On the other hand, if none of its children is true, it will return false.

Sequencer. A sequencer task will return immediately with a failure (false) is one of its children fails. It is denoted with an arrow inside a circle.

$$[\![\textcircled{\rightarrow}]\!] = out(f(c)) = \begin{cases} \text{True} & \text{if } i = \text{True} \quad \forall i \in c \\ \text{False} & \text{otherwise} \end{cases} \tag{5}$$

The meaning of a sequencer task is expressed by equation 5, if one of the children is false, it will return false. On the other hand, if none of its children is true, it will return true.

In the cases of both the sequencer and selector task, the evaluation of the children is deterministic, such as the order is respected. It is also possible to use non-deterministic composition task. The random sequencer and selector are examples of this.

Random Selector. The random selector acts exactly as the equation 4 describes it, although every time the node is evaluated, it rearranges randomly the order of c. It is denoted with a tilt followed by a interrogation point: $\textcircled{\sim ?}$.

Random Sequencer. The random selector acts exactly as the equation 5 describes it, although every time the node is evaluated, it rearranges randomly the order of c. It is denoted with tilted arrow: $\textcircled{\sim\rightarrow}$.

Decorator. The name decorator refers to the design pattern "decorator" that wraps an another class modifying its behaviour. The decorator task has the same interface as any composite task. In the context of BT, a decorator has only one child and modifies it in some matters.

$$\text{Decorator Task} =< f(e), e, out > \tag{6}$$

The definition of a decorator is presented in equation 6. It is important to note that e here refers to one child, that is why it is denoted differently than in equation 3, *out* is a boolean variable based on the success and failure of $f(e)$ and $f(e)$ will take into consideration the child node. For this project we will only consider one decorator task: Inverter task.

Inverter. The inverter simply return the inverse of its child. It is denoted with a interrogation point inside a circle.

$$[\![\textcircled{!}]\!] = out(f(e)) = \begin{cases} \text{True} & \text{if } e = \text{False} \\ \text{False} & \text{otherwise} \end{cases} \tag{7}$$

The meaning of a sequencer task is expressed by equation 7, if the child returns true, the inverter task returns false, otherwise it is true.

Behaviour Trees. In general BT are form of task nodes where the leafs are action or condition tasks and the branches are composite tasks such as selector task. In this case we define a behaviour tree as follow:

$$\text{Behaviour Trees} =< e > \tag{8}$$

In equation 8, e represents one task node. The task node is evaluated at every update of the simulation. In general, the output of the tree has no influence on the simulation. Although, someone could use the output of the tree to influence the simulation. Moreover, with this structure it is possible to build complex behaviour.

Example

In order to fully grasp the power of BT, we are going to explore a small example step by step. The example represents the behaviour of a guard in a video games. We are interested in the case where the guard sees an enemy. In this case, she will engage combat using a randomly choose combat tactic if her health level is high enough. In the case where her health level is not high enough, she will try to flee. If she cannot feel, she will fight.

In Figure 1 we can see the behaviour presented translated into a BT. The evaluation starts at the sequencer at the top (1). In sequencer and selector task, the children are evaluated from left to right, or first in first out. The first child to be evaluated is (2). If the guard can see an enemy it returns true, if there is no enemy, it returns false which cause the sequencer at (1) to return false as well. In the case where there is an enemy, node (3) is evaluated. If the health level of the guard is low (4), then it will try to run away from combat (6). In the case where the guard does not have enough health (4) and can flee (6), the sequencer (4) will return true to (3), which will cause (3) to return false to (1). In this case, the guard does not attack the enemy and flee. Moreover when the health level is high enough, (4) will return false to (5) which will return false to (3) which will return true to (1), which cause the guard to attack the enemy. When (7) is evaluated, it randomly changes the order of (8), (9) and (10). Once the order is changed, all the actions are picked until one returns true. In the case where none returns true, it returns false.

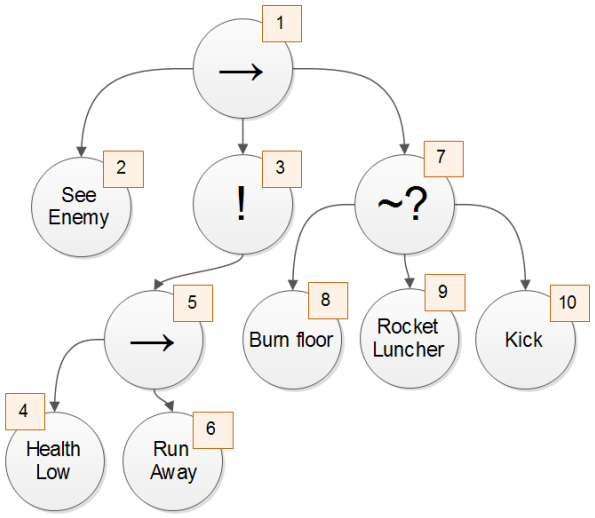


Figure 1: Behaviour trees example

As you can see the BT translates well the behaviour presented before. BT are powerful and easy to use, since the understanding is pretty straight forwards, it is possible to give a set of actions and conditions to designer (non-programmer) in order to build interesting behaviours for games. Moreover, games are not the only place where BT are useful, they are also used in robotics or simulating AI.

Implementation

Behaviour trees is used bye different fields such as robotics, AI for video games, human behaviour simulation, etc. In our case, we are particularly interested in building AI for games. Therefore, BT were implemented inside *Unity 3.5.1*. *Unity* is a free integrated authoring tool for creating 3D video games. The development tool runs on both *Windows* and *Mac OS*. The compiled *Unity* game runs on *Windows, Mac OS, Xbox 360, Playstation 3, Wii, iPad, iPhone* and *Android*. Also, *Unity* uses *C#* and *JavaScript* as programming language.

For this project a simple game was develop in *Unity* using *C#*. Figure 2 is a screen shot of the game/simulation running. The game is quite simple, the red cylinder, AI, has to reach the blue cube while avoiding/shooting at

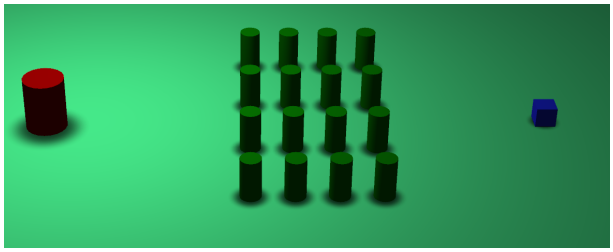


Figure 2: A screen shot of the game

the green cylinders. The green cylinders, enemy, are not friendly towards the AI. If the AI gets too close to or shoot one enemy, the enemy will chase the AI. When the enemy touches the AI, the AI looses some health. As defence, the AI can shoot at them, three bullets are needed to take an enemy down.

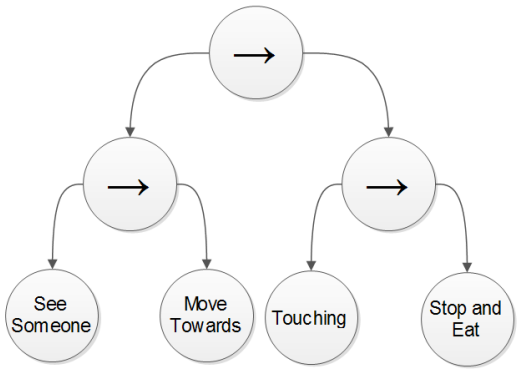


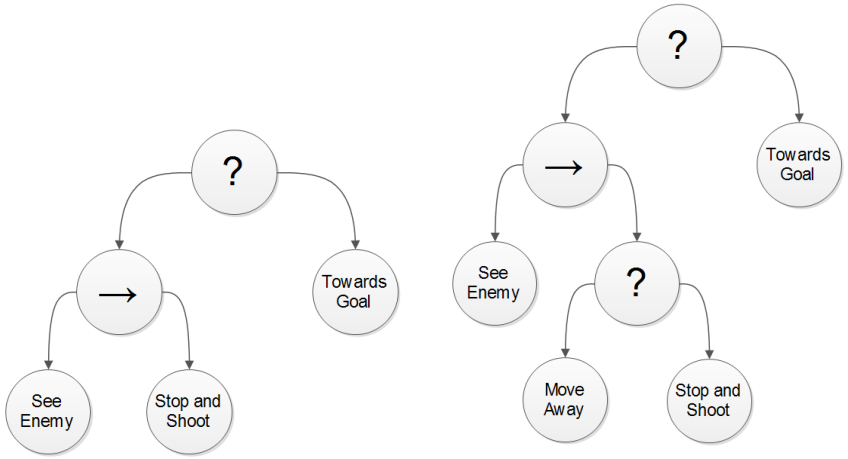
Figure 3: The enemy behaviour trees

The AI and enemies behaviour are implemented inside BT. They used the internal and external nodes presented in the previous section. The AI's behaviour trees will be discuss in the following section.

Figure 3 is the enemy behaviour tree. When the enemy sees someone, it will start chasing it by moving towards it in a straight line. When it collides with that person it stops moving and "eat". When an enemy receives a shot from the AI, it adds the AI to its list of things it could see. This causes the enemy to chase the AI as it can see it.

Experiment

For this project two different BT were designed, the first one simulates a killer, she does not try to dodge anything. She goes for the goal, and if something is in the way, she shoots at it. The second AI is more subtle, she will try to avoid any hazard and try to reach the goal using sophisticated dodging.



(a) Killer (b) Dodger

Figure 4: Behaviour Trees

Killer. As expressed, the killer goes directly at the goal and shoot. In Figure 4a we can see the actual BT used to represent that behaviour. When the killer sees an enemy, she will shoot at it. Otherwise she is moving towards her goal. The killer is always shooting at the closest enemy.

Dodger. The second developed AI is the dodger, she will try to avoid combat as much as possible. In Figure 4b is the BT, as you can see, she will move around the enemy in order to find a safe route to the goal. If it is impossible to move away from the enemy, she will start shooting at the enemy. If no one is encounter, she is moving towards the goal.

Metrics

In order to measure the output of both the AI, an observer was implemented to gather information about the simulation. In this case we are

interested in defining performance metrics that will give a profile to the AI, we used five metrics.

1. Time: how long did it take for the AI to reach the item
2. Bullet used: how many bullets were used in order to reach the item
3. Average health: average health level for the simulation
4. Enemies killed: how many enemies were killed
5. Tree information: a percentage of usability given to every node in the tree

The tree information gives a percentage rate for every note, this percentage represents the rate the node was visited. Those metrics are used to evaluate the performance and profile the AI.

Results

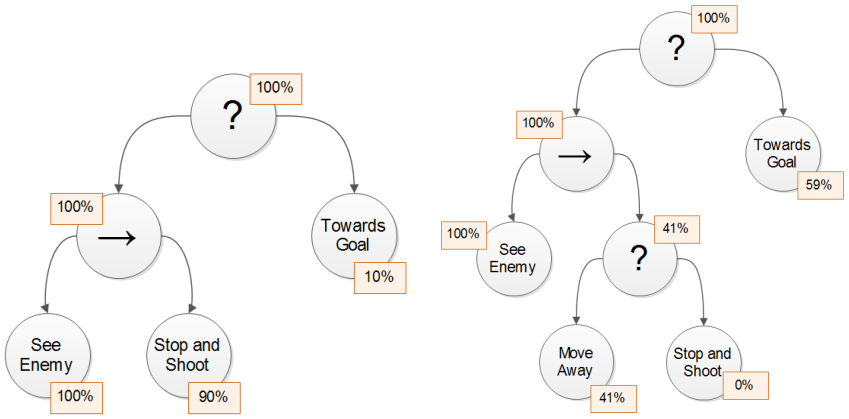
The simulation was run inside the game structure presented in Figure 2. In total, 16 enemies are there to take down. Both AIs were tested over multiple simulations. In Table 1 we can see that both BTs are really different

Table 1: Simulation result

	Killer	Dodger
Time	152	19.57
Bullet used	50	0
Average Health	67.3	100
Enemies Killed	16	0

in term of outcomes. First, the time spent on the level is drastically different, the killer for sure will take longer as she has to move through the hoard of enemy that you can see in Figure 2 in order to reach the item. Shooting enemies takes a little while, on the other hand the dodger does not shoot anyone, she goes around. As you can see the dodger did not have to kill anyone, as she was not set into a trap or blocked against a wall. Because the world does not have any walls, the dodger does not have to kill anyone.

In Figure 5, you can see the tree information from the simulations. In Figure 5a, you can see that 90% of the updates is spent shooting and only 10% to move towards the item. On the other hand, in Figure 5b, 41% of the time is spent trying to avoid the enemies and 59% of the time moving towards the item. It is interesting to see that the dodger does not use *Stop*



(a) Killer (b) Dodger

Figure 5: Tree information

and Shoot, as expressed before it cause by the world not having any walls. She goes around them without getting stock. In this case, this node is useless, this metric allows us to be critic towards our design. Since it is never used, maybe it is best not to have it. This metric in particular locates useless tasks and give an input about was to do with it.

Moreover, these metrics allow us to build a profile for any design. As we can see the killer is slow and will make sure her path is clear before moving forwards. On the other hand, the dodger is fast but does not kill anyone. These profiles are interesting in order to design *gameplay* in games. Imagine the player has to send three units to recover the item, she decided when to send them, maybe she will try to send a killer, a dodger and a killer again, or any arrangement. Without those metrics, you can guess how the BT will interact with the simulation, but it will not be possible for you to make any real conclusions about their behaviours.

3. Conclusions

In this report a description of Behaviour Trees were presented. The formalism is really useful to model complex behaviour using only *task* such actions and conditions. The formalism was implemented inside a simple game simulation, which allowed us to develop environment oriented metrics in order to capture the essence of the AI. Moreover, there exists more tasks node, such as *parallel selector* which introduces the idea of parallel task execution.

This task shows how specialize BT can become and how creating a task is strongly adaptive towards the goal to accomplish. It would be interesting to develop more about this adaptivity and how the creation of a new task could influence the comportment of BT.

References

[1] Alex J. Champandard. Getting Started with Decision Making and Control Systems. *AI Game Programming Wisdom 4*, 2008.

[2] B. Iske and U. Ruckert. A methodology for behaviour design of autonomous systems. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 2001.

[3] Damian Isla. GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI. `http://www.gamasutra.com/view/feature/2250/gdc_2005_proceeding_handling_.php`, 2005. [Online; accessed 28-April-2012].

[4] Chong-U Lim. An AI Player for DEFCON: an Evolutionary Approach Using Behaviour Trees. *Imperial College London*, 2009.

[5] Ian Millington and John Funge. Artificial Intelligence for Games, Second Edition. 2009.

[6] Ugo Di Profio Yukiko Hoshino, Tsuyoshi Takagi and Masahiro Fujita. Behavior Description and Control Using Behavior Module for Personal Robot. *ICRA*, 2004.