

Modeling Battle Drills for Computer-Generated Forces using Behavior Trees

Per-Idar Evensen, Håvard Stien, Dan Helge Bentsen
Norwegian Defence Research Establishment (FFI)
Kjeller, Norway
per-idar.evensen@ffi.no, havard.stien@ffi.no, dan-helge.bentsen@ffi.no

ABSTRACT

Modeling realistic human behavior, including decision-making and creativity, is the hardest and most complex challenge in combat simulation. Behavior trees (BTs) are a relatively new and increasingly popular approach for developing behavior models for artificial intelligence (AI) and intelligent agents. The approach has become especially popular for creating behaviors for non-player characters (NPCs) in computer games, robots, and autonomous vehicles. The first high-profile computer game that used BTs was Halo 2 from Bungie Software, which was released in 2004.

BTs are represented as directed trees with a hierarchy of control flow nodes and task nodes that control the behavior of an agent. The control flow nodes are interior nodes and contain decision logic for flow control. The task nodes are leaf nodes and contain conditional tasks that test some property in the simulated environment (or the real world in case of robots and autonomous vehicles) or action tasks that alter the state of the simulation (or the real world) in some way.

What makes BTs so powerful is their composability and modularity. Task nodes and control flow nodes are composed into sub-trees that represent more complex actions, and these actions can be composed into higher level behaviors. Task nodes and action sub-trees can be reused, and different sub-trees can be developed independently of each other.

In this paper we will give an introduction to BTs based on available literature and discuss the possibilities and limitations of employing this modeling technique for creating behavior models for computer-generated forces (CGF) in combat simulations. Furthermore, we will give a concrete example of how to create a BT from a textual description of a battle drill (from a field manual), and provide tips and tricks on how to create BTs in general. Finally, we will summarize our experiences from working with BTs.

ABOUT THE AUTHORS

Per-Idar Evensen is a Senior Scientist at the Norwegian Defence Research Establishment (FFI), where he has worked since 2006. His research work focuses on modeling and simulation (M&S) of land force operations at different levels for experimentation and analysis purposes, and he has a special interest in simulation systems based on game technology. He has a Master of Science degree in Computer Science from the University of Oslo in 2004.

Håvard Stien is a Research Engineer at the Norwegian Defence Research Establishment (FFI), where he has worked since 2016. His work consists of developing new capabilities for military simulation systems. He has a bachelor's degree in Computer Science from UiT The Arctic University of Norway in 2016.

Dan Helge Bentsen is a Chief Research Engineer at the Norwegian Defence Research Establishment (FFI), where he has worked since 2009. He is a 2nd Lieutenant (Reserve) in the Norwegian Army, and has previously served as a tank commander. He has a bachelor's degree in Multimedia Technology and Design from the University of Agder in 2008. His work mainly focuses on the use of modeling and simulation (M&S) tools to evaluate the combat effectiveness of different land force structures and components.

Modeling Battle Drills for Computer-Generated Forces using Behavior Trees

Per-Idar Evensen, Håvard Stien, Dan Helge Bentsen
Norwegian Defence Research Establishment (FFI)
Kjeller, Norway
per-idar.evensen@ffi.no, havard.stien@ffi.no, dan-helge.bentsen@ffi.no

INTRODUCTION

Modeling realistic human behavior, including decision-making and creativity, is the hardest and most complex challenge in combat simulation (Thorpe, 2010). Behavior trees (BTs) are a relatively new and increasingly popular approach for developing behavior models for artificial intelligence (AI) and intelligent agents. The approach has become especially popular for creating behaviors for non-player characters (NPCs) in computer games, robots, and autonomous vehicles. The first high-profile computer game that used BTs was Halo 2 from Bungie Software (Isla, 2005), which was released in 2004.

Finite state machines (FSMs) have long been the most dominant technique for creating behavior models for computer generated forces (CGF) in military simulations (Jones, 2011; Abbott et al, 2010), but BTs are now also starting to become a popular technique for developing behavior models for automated and semi-automated constructive entities in military simulation systems. FSMs are well suited for implementing well-defined doctrinal behavior of limited complexity (Abbott et al, 2010). However, the problem with FSMs is that they tend to become very complex and unmanageable. The reason for this is that the number of states increases exponentially with the number of non-mutually exclusive behaviors that are represented (Papelis & Madhavan, 2010), and as the number of states increases, the number of possible transitions between the states also increases exponentially (Buede et al, 2013).

BTs have some similarities to hierarchical FSMs, but the key difference is that their main building blocks are *tasks* rather than *states*. BTs are represented as directed trees¹ with a hierarchy of control flow nodes and task nodes that control the behavior of an agent. The control flow nodes are interior nodes and contain decision logic for flow control. The task nodes are leaf nodes and contain conditional tasks that test some property in the simulated environment (or the real world in case of robots and autonomous vehicles) or action tasks that alter the state of the simulation (or the real world) in some way (Evensen & Bentsen, 2016).

What makes BTs so powerful is their composability and modularity. Task nodes and control flow nodes are composed into sub-trees that represent more complex actions, and these actions can be composed into higher level behaviors (Millington & Funge, 2009). Task nodes and action sub-trees can be reused, and different sub-trees can be developed independently of each other.

BT editors with graphical user interfaces (GUIs) enable users, for example military officers, to create modular behavior models without needing programming skills. Furthermore, to reduce complexity and ensure readability of the graphical model of large BTs, they can be decomposed into smaller sub-trees. Examples of AI engines for military simulation systems that use BTs are Virtual Battlespace (VBS) Control from Bohemia Interactive Simulations (BISim) and MASA Life from MASA Group.

First, in this paper, we briefly describe the background for this work. Secondly, we give an introduction to BTs, and look at the process of developing BTs. Then, we discuss the advantages and limitations of employing this modeling technique for creating behavior models for computer-generated forces (CGF) in combat simulations, and look at some of the possible extensions of BTs. After this, we describe how we are using BTs to build a library of behavior models for the most important battle drills for mechanized infantry platoons and give a concrete example of how to create a BT from a textual description of a battle drill. Finally, we summarize our experiences from working with BTs and outline our plans for further work.

¹ Strictly speaking, a BT is a *directed acyclic graph* (DAG) since the same node or sub-tree can be used several places in the structure, and a node can thus have more than one parent.

BACKGROUND

At the Norwegian Defence Research Establishment (FFI) we conduct constructive entity-level simulations of battalion to brigade level operations for analysis and experimentation purposes. One of the main research questions we are investigating is *how to increase combat effectiveness* in land force operations, and as part of this work we need to assess and compare the performance of different land force structures, which may vary with regard to: *composition of material and equipment, tactical organization, and operational concept*. Our simulation experiments are conducted as what can be described as *simulation-supported, two-sided* (blue and red) *wargames*, where military officers participate as players/operators on both sides (Evensen & Bentsen, 2016; Evensen et al, 2017).

We have identified two main factors that have the potential to improve the fidelity of our constructive simulations: (1) *increased terrain resolution*, and (2) *better tactical artificial intelligence* (AI) that can exploit this terrain (Evensen & Bentsen, 2016; Evensen et al, 2017). For example, we expect that these two factors will result in more realistic detection and engagement distances in our constructive simulations (Evensen et al, 2017). Furthermore, since we are using constructive simulation to experiment with utilization of new technologies and new concepts on the battlefield, we need an easy way to make changes to behavior models for experimenting with different tactical behavior of entities and platoons. The composability, modularity, and readability of BTs make this modeling technique very suitable for our use.

We are composing a simulation system where the ground-to-ground combat entities are simulated in VBS from BISim, and the air and air defense entities are simulated in VR-Forces from VT MAK. All the constructive, semi-automated entities are controlled from a web-based graphical user interface (GUI) (Evensen et al, 2017). The entities simulated in VBS will use behavior models developed in VBS Control. VBS Control is a new framework for AI in VBS, based on BTs. To get better tactical AI in our simulations, we are currently building a BT-based library of behavior models for the most important battle drills for mechanized infantry platoons.

INTRODUCTION TO BEHAVIOR TREES

BTs are a relatively new technique for developing behavior models. Nevertheless, BTs have now reached a certain maturity and have been treated in at least two Game AI textbooks (Millington & Funge, 2009; Rabin, 2013). Furthermore, a number of BT tutorials can be found on the Internet. Some of the available descriptions of BTs differ slightly, and a suggested unified framework for BTs was published in 2014 (Marzinotto et al, 2014).

Structure

BTs are graphically represented as *directed rooted trees*. Directed rooted trees are composed of *nodes* and directed *edges* connecting the nodes. Trees cannot contain any cycles. For a pair of connected nodes, the outgoing node is called the *parent*, and the incoming node is called the *child*. A parent node can have one or more children. Rooted trees have one parentless node that is called the *root*. Nodes without children are called *leaves*. In BTs the edges are directed away from the root and towards the leaves. BTs are usually drawn with the root at the top, and the children are usually ordered from left to right.

Traversal

A BT represents all the possible courses of action an agent can take, and a path from the root to one of the leaf nodes typically represents one possible course of action (Millington & Funge, 2009). BTs are essentially traversed in a *depth-first* manner (from left to right). Starting from the root, the leftmost children are visited first until a leaf node is reached, before backtracking and visiting the next child (to the right) of the nearest node with more children.

An AI engine will usually traverse a BT from the root for each *simulation step* or *tick*, executing each node down the tree. The simulation step or tick may correspond to the frame rate of the simulation, but it can also be longer. The simulation step represents the maximum time it will take for the behavior model to react to a change in the environment, so for a model representing human behavior, it should not be longer than a typical human reaction time (usually between 0.2 and 0.4 seconds).

During the traversal of a BT each child node will return one of the following three status values to its parent:

1. *Success*: The node achieved its goal
2. *Failure*: The node failed
3. *Running*: The node did not finish its execution within the current simulation step and is still running

The running status is typically returned by tasks that take some time to complete, e.g. moving from one place to another.

Each time a BT is executed it either finishes by returning success or failure back to the root node, or the execution ends up in a node representing a task that takes some time to execute and thus is still running at the end of the simulation step. At the next simulation step, the AI engine can either be designed to continue the BT execution at the leaf node that returned running, or most commonly, to restart the BT execution at the root. Starting at the root at each simulation step allows the behavior model to be more *reactive* to context changes, which is an important property in dynamic environments. Of course, if there are no context changes, the BT execution will end up in the node that returned running at the previous simulation step and continue the execution of this task.

Types of Nodes

The main categories of nodes in a BT are *control flow nodes* (or *composite nodes*), *task nodes* (or *execution nodes*), and *decorator nodes*. In addition, a BT can have *reference nodes*, which are just references to a sub-tree.

The control flow nodes are the interior nodes in a BT, and they always have one or more children. There are three types of control flow nodes: *selector nodes*, *sequence nodes*, and *parallel nodes*. The task nodes are the leaf nodes in a BT. There are two types of task nodes: *condition nodes* and *action nodes*.

Selector Nodes

A selector² node will start to execute each of its children from left to right and return success as soon as one of the children returns success. If none of the children returns success, the selector node will return failure. If the child that is currently being executed returns running at the end of a simulation step, the selector node will return running. Figure 1 shows the graphical representation of a selector node with N children. The selector node is denoted by a question mark.

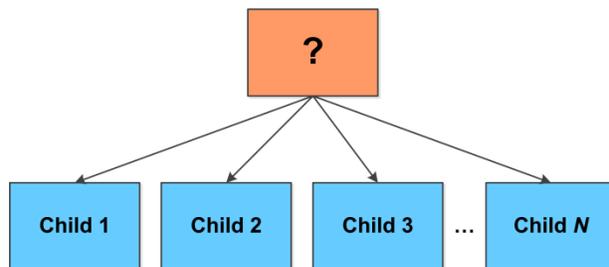


Figure 1. Graphical representation of a selector node with N children

Selector nodes are typically used when a set of actions represents alternative ways of reaching a goal. Some BT implementations also define non-deterministic or random selector nodes, where the children are executed in a non-deterministic order.

Sequence Nodes

A sequence node will start to execute each of its children in sequence from left to right and return failure as soon as one of the children returns failure. If all the children return success, the sequence node will return success. If the child that is currently being executed returns running at the end of a simulation step, the sequence node will return running. Figure 2 shows the graphical representation of a sequence node with N children. The sequence node is denoted by a rightwards arrow (\rightarrow).

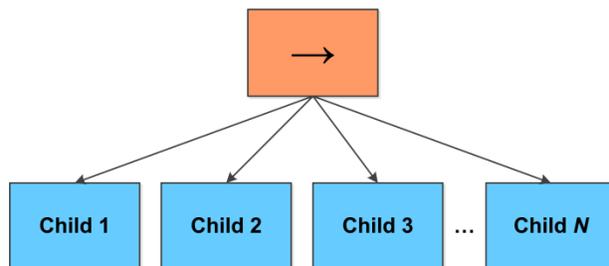


Figure 2. Graphical representation of a sequence node with N children

² Selector nodes are sometimes called fallback nodes.

Sequence nodes are typically used when a set of actions needs to be carried out in a particular order. Some BT implementations also define non-deterministic sequence nodes, where the children are executed in a non-deterministic order.

Parallel Nodes

A parallel node will execute all of its children in parallel. If one child returns failure, the parallel node will return failure and terminate the execution of all the other children. If all the children complete successfully, the parallel node will return success. If none of the children have returned failure, and one or more children return running at the end of a simulation step, the parallel node will return running. Figure 3 shows the graphical representation of a parallel node with N children. The parallel node is denoted by rightwards paired arrows (\Rightarrow).

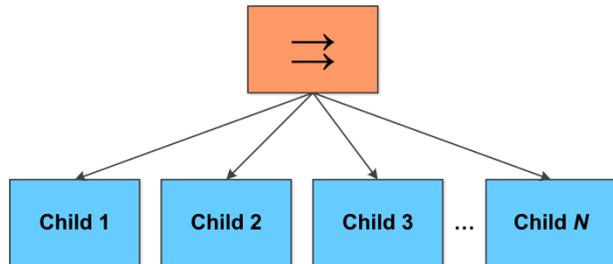


Figure 3. Graphical representation of a parallel node with N children

Parallel nodes are typically used when a set of actions can be carried out at the same time. Some BT implementations define parallel nodes with different criteria for success, for example return success if more than M out of N children return success, or only return failure if all of the children return failure.

Condition Nodes

A condition node checks if a given condition within the simulated environment (or the real world) is fulfilled. If the condition is fulfilled it returns success, otherwise it returns failure. A condition task will always complete within a simulation step, so this task will never return running. Figure 4 (to the left) shows the graphical representation of the condition node. The “*Condition*” text label will typically be a description of the condition.



Figure 4. Graphical representation of a condition node (to the left) and an action node (to the right)

Action Nodes

An action node performs an action, which alters the state of the simulated environment (or the real world) in some way. If the action was completed the action node will return success, and if the action could not be completed the action node will return failure. Running will be returned if the action was not finished within the current simulation step. Figure 4 (to the right) shows the graphical representation of the action node. The “*Action*” text label will typically be a description of the action.

Decorator Nodes

A decorator node is a special type of node that has only one child and modifies the behavior of the child according to some predefined rule. Examples of decorators are inverter nodes that invert the result from the child (i.e. success to failure and failure to success), succeder nodes that always return success, and repeater nodes that repeat the execution of the child a specific number of times or until a given condition is fulfilled. A BT can have several subsequent decorators. Figure 5 shows the graphical representation of the decorator node. The “*Decorator*” text label will typically be a description of the rule of the decorator.

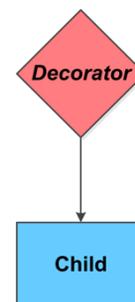


Figure 5. Graphical representation of a decorator node and its child

Data Store

What makes BTs highly modular and flexible, is the concept of having a single common interface for all nodes. This allows that any task or sub-tree can be placed at any position in a BT. For this to work, the data used by the BT needs

to be decoupled from the interface between the nodes. This is usually done by having an external data store, called a *blackboard*, for all the data that the BT needs.

Performance

The performance of a BT will depend on the complexity of the task nodes. If we assume that the task nodes (leaf nodes) are of $O(1)$ in both performance and memory usage, the average performance for a BT will be of $O(\log n)$ and the memory usage for a BT will be of $O(n)$, where n is the number of nodes in the BT (Millington & Funge, 2009).

Example: Moving Into a Room

Figure 6 shows an example of a simple BT for moving into a room. When this BT is executed, the selector node will first try to execute the left child, which is a sequence node. The sequence node will first execute the condition, which checks if the door is open. If the door is open, the sequence node will execute the next child, which is an action node that moves the agent into the room. If the door is closed, the sequence node will return failure, and the selector node will try to execute the right child, which is also a sequence node. This sequence node will try to execute the following three action nodes: “Move to door”, “Open door” and “Move into room”. If any of these three action nodes fail, the whole BT will return failure.

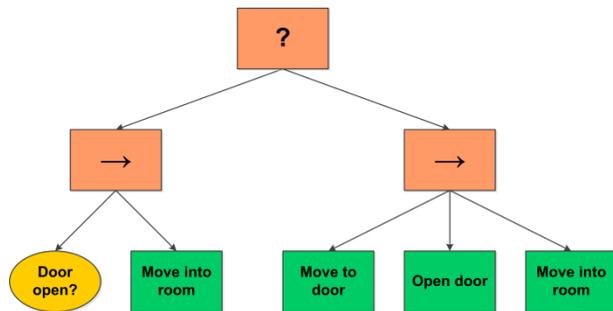


Figure 6. Simple BT for moving into a room

DEVELOPING BEHAVIOR TREES

Development of BTs is an iterative process, where we typically start with a simple BT and then make it more complex by adding more and more branches of alternative courses of action for achieving a goal. Generally, in a BT the left branch of the tree (starting from the root) will contain the high-priority behaviors, while the right branch of the tree will contain the low-priority behaviors. The default or unconditional behavior will therefore be found at the far right side of a BT.

An important question when designing a BT is what functionality to address in the BT structure, and what functionality to take care of inside the action nodes (Colledanchise & Ögren, 2018). Generally, most modularity is achieved if each task can be broken into the smallest parts that can usefully be composed (Millington & Funge, 2009). However, a BT that is too fine grained may be unnecessarily complex (Colledanchise & Ögren, 2018). As a rule of thumb, a BT should be decomposed into the smallest action nodes which do not have sub-parts that are likely to be usable as stand-alone actions in other parts of the BT.

Developing and editing BTs are mostly done using visual editors, but BTs can also be automatically generated from examples of expert behavior by using machine learning techniques (Robertson & Watson, 2015; Colledanchise et al, 2016).

ADVANTAGES, LIMITATIONS AND POSSIBLE EXTENSIONS OF BEHAVIOR TREES

In AI research, BTs are classified as a simple form of planning referred to as reactive planning (Millington & Funge, 2009). To be reactive means to be able to quickly react to changes. Behavior models using reactive planning techniques are suitable for handling highly dynamic and unpredictable environments, such as a modern battlefield.

From a theoretical point of view, there is no difference between BTs and FSMs regarding what kind of behavior that can be modeled (Marzinotto et al, 2014). In practice however, whereas FSMs tend to become impractical and unmanageable as the complexity of the behavior model grows, complex BTs are much easier to maintain and extend.

Advantages

The most important advantages with BTs can be summarized as follows:

- BTs are highly *composable*. Composability means the ability to combine components into various combinations for building different systems.
- BTs are highly *modular*. That a system is modular means that it can be subdivided into modules (tasks or sub-trees for BTs), and that any module in the system can be replaced by any other module. This also means that the modules can be developed independently of each other.
- BTs are *reactive*, which means that they can react quickly to changes.
- BTs are *human readable* and can be created by *visual editors* with GUIs.
- BTs are suitable for *automatic generation*, for example by using machine learning techniques.

All these properties make BTs well suited for developing human behavior models of moderate complexity for semi-automated forces (SAF) in constructive simulations.

Limitations

The most important limitations of BTs can be summarized as follows:

- BTs are poor at modeling the uncertainty in situations where there are multiple valid options to choose from (Merrill, 2013).
- It is somewhat cumbersome to represent typical state-based behavior using BTs (Millington & Funge, 2009).
- There are limitations on how large and complex behavior models can be when using BTs. For very large BTs, the cost of having to execute the whole tree from the beginning for each simulation step will eventually cause performance issues, especially in simulations with a high number of constructive entities.

Possible Extensions

To overcome the limitations of standard BTs, several extensions have been proposed.

The first limitation is mainly related to the properties of the selector node. One simple way to vary the order in which the selector node executes its options, is to introduce non-deterministic selector nodes. This will lead to more unpredictable behavior. A more comprehensive way to address this limitation is to combine BTs with utility-based decision-making, as suggested in (Merrill, 2013). Here, a utility selector node is introduced, which when executed first queries all of its children for a utility value, and then uses these values to determine in which order the children will be executed. The calculation of utility values must typically be done by the task nodes, and then propagated up the tree structure. This method will however require additional computing power for each simulation step.

A simple solution which will overcome the second limitation is to combine BTs with FSMs (Millington & Funge, 2009). Instead of having one large BT representing all the possible behaviors of an agent, it will then be possible to have different context-sensitive BTs for each of the states the agent can be in, and a simple FSM on top that handles the transitions between the states.

The limitation regarding performance issues with large BTs is harder to solve without compromising the reactivity of the BTs. To avoid unwanted re-execution of all control flow nodes for each simulation step, control flow nodes with memory have been suggested. The control flow nodes with memory will then remember what value each child has returned, and avoid re-executions of the children until the whole control flow node has returned success or failure (Colledanchise & Ögren, 2018). This approach will, however, make the BT less reactive, and will therefore not be suitable for use in dynamic and unpredictable environments.

BUILDING A LIBRARY OF BEHAVIOR MODELS OF BATTLE DRILLS

As mentioned, we are currently building a BT-based library of behavior models of the most important battle drills for mechanized infantry platoons, including dismounted soldiers and combat vehicles like infantry fighting vehicles (IFVs) and main battle tanks (MBTs). The behavior model library will have a hierarchical structure with models of battle drills for entities, squads, and platoons for dismounted soldiers, and entities and platoons for combat vehicles. Human operators will give orders to the semi-automated forces (SAF) at the squad or platoon level, but we want the entities to be completely autonomous within a squad for dismounted soldiers and within a platoon for combat vehicles. In the future, we envisage building behavior models for a set of more generic battle drills at the company level, so that more general orders can be given at this level. It is a goal that one operator should be able to control an entire battalion of maneuver forces.

The workflow for creating the behavior models follows a bottom-up approach, where we first model a set of low-level actions that the simulated entities should be able to execute. These actions are typically represented as action nodes. From the low-level actions we compose sub-trees representing more complex actions or tasks for the individual entities, and these are further used to build BTs of battle drills for the entities. Typically, much work is required to create the first BTs, but as the library grows, it is more and more likely that it is possible to reuse already built sub-trees when building new BTs.

In addition to the BTs for the individual entities, we build BTs for battle drills at the unit level for squads and platoons. The unit level behavior models coordinate the behavior of the individual entities and issue orders (i.e. assigns BTs) to the entities. The unit level behavior models thus represent the leading element of the unit. Messages are used for communication between the unit level BT and the entity level BTs, for example for sending orders from the unit level BT to the entity level BTs and sending reports from the entity level BTs to the unit level BT. The same principle is used to build additional levels of behavior models, for example for infantry we have behavior models at the entity level, squad level and platoon level. In principle there are no limitations on the number of levels of BTs that can be modeled, and in the future we also plan to build BTs at the company level.

To get an overview of the different tasks a unit should be able to conduct, it is possible to use so called Universal Task Lists (UTLs). Detailed descriptions of tasks and battle drills can often be found in field manuals and by consulting subject matter experts (SMEs) and officers. At the unit level we build BTs corresponding to the orders that can be issued by the human players/operators. Figure 7 shows an example of orders that can be issued to generic combat vehicle platoons. The orders are categorized as “Offensive”, “Defensive” or “Move”. For our simulation-supported, two-sided wargames we need to create models of battle drills for both blue and red forces, since they usually follow different doctrines.

Example: Model of a Contact Drill for an Infantry Squad

In this subsection we will first give a short description of a battle drill for enemy contact for a dismounted infantry squad. The contact drill has been taken from an unclassified Norwegian field manual for dismounted infantry squads (Norwegian Army, 2014). After this we will look at the modeled squad level and entity level BTs for the contact drill.

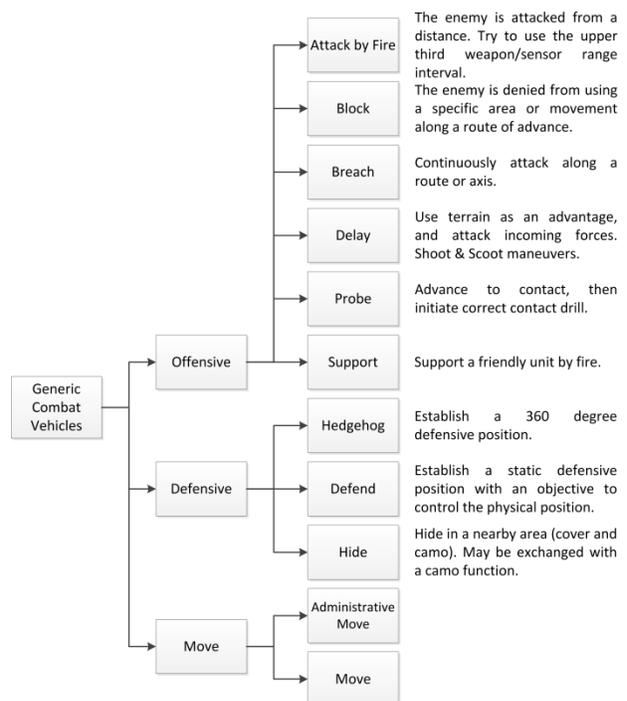


Figure 7. Orders for combat vehicle platoons

Description of contact drill:

1. Soldier 1 (scout) discovers the enemy in front, fires a burst and keeps firing rapid single shots while shouting “Contact front!” and seeking cover. This is illustrated in Figure 8 (a).
2. Soldier 2 (squad leader) moves to the right (or left), provides covering fire, and shouts “Covering!”. At the same time the rest of the squad forms a line to the left (or right) of Soldier 2. This is illustrated in Figure 8 (b).
3. As soon as Soldier 1 is covered by Soldier 2, he or she starts to withdraw to a new position. Soldier 1 and Soldier 2 conduct backwards leap-frogging, while the others find positions and fire a burst followed by single shots. This is illustrated in Figure 8 (c).
4. Soldier 1 and Soldier 2 pass Soldier 3 on their way backwards, and the squad will split into two fireteams. Fireteam 1 moves collectively backwards to new positions while Fireteam 2 provides covering fire. Afterwards, Fireteam 2 moves backwards to new positions while Fireteam 1 provides covering fire. This is illustrated in Figure 8 (d).
5. The leap-frogging continues until they find a position where they can break off contact, and the squad can find cover and concealment from the enemy. The squad then forms a line and moves towards the cover. This is illustrated in Figure 8 (e).

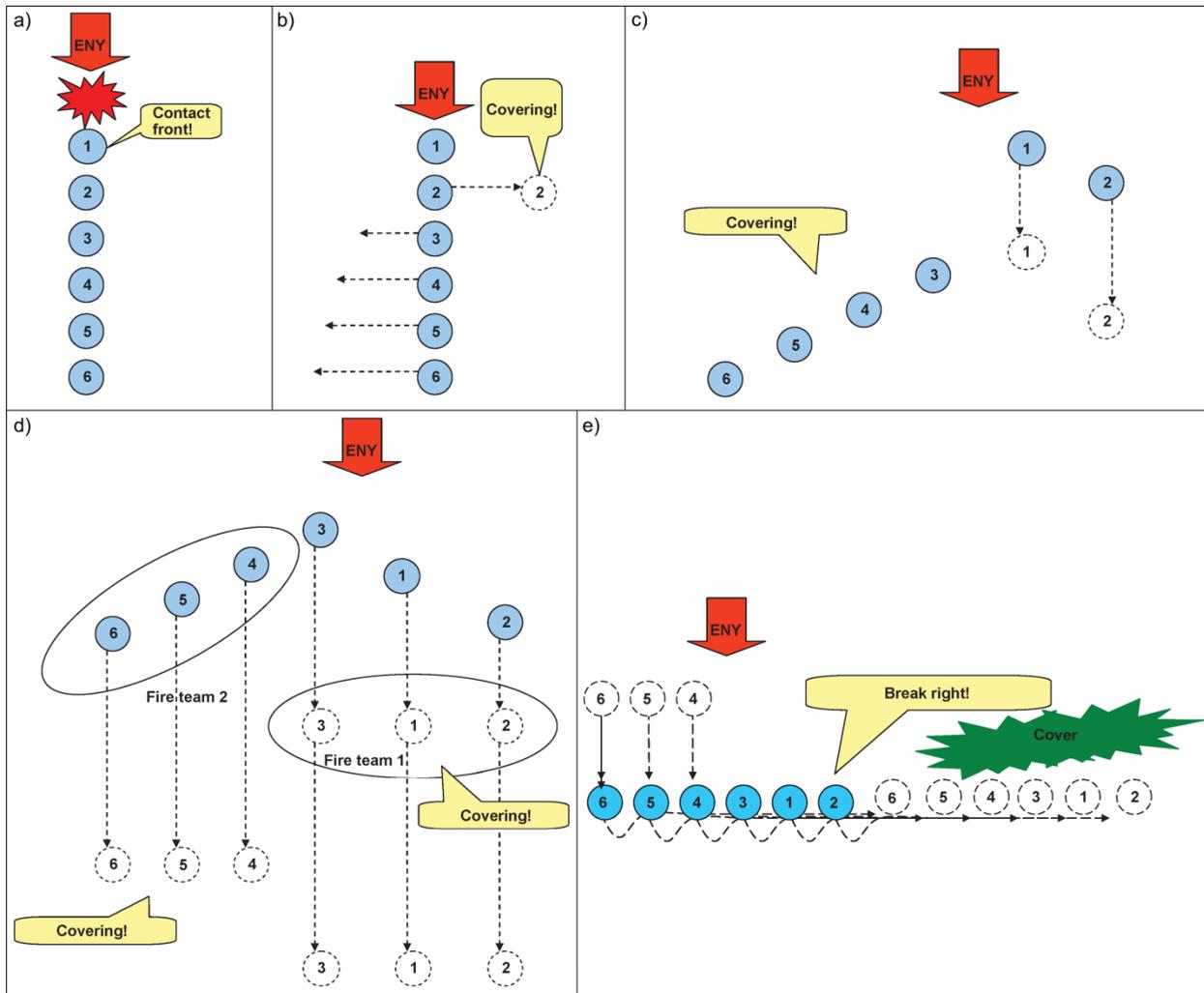


Figure 8. Battle drill for enemy contact for a dismounted infantry squad (Norwegian Army, 2014)

Modeled Behavior Trees for Contact Drill

Figure 9 shows the modeled BT for the squad, and Figure 10 shows the modeled BT for the first soldier in the squad (Soldier 1). The squad BT for the contact drill will typically be used as a sub-tree in a BT for a squad move order.

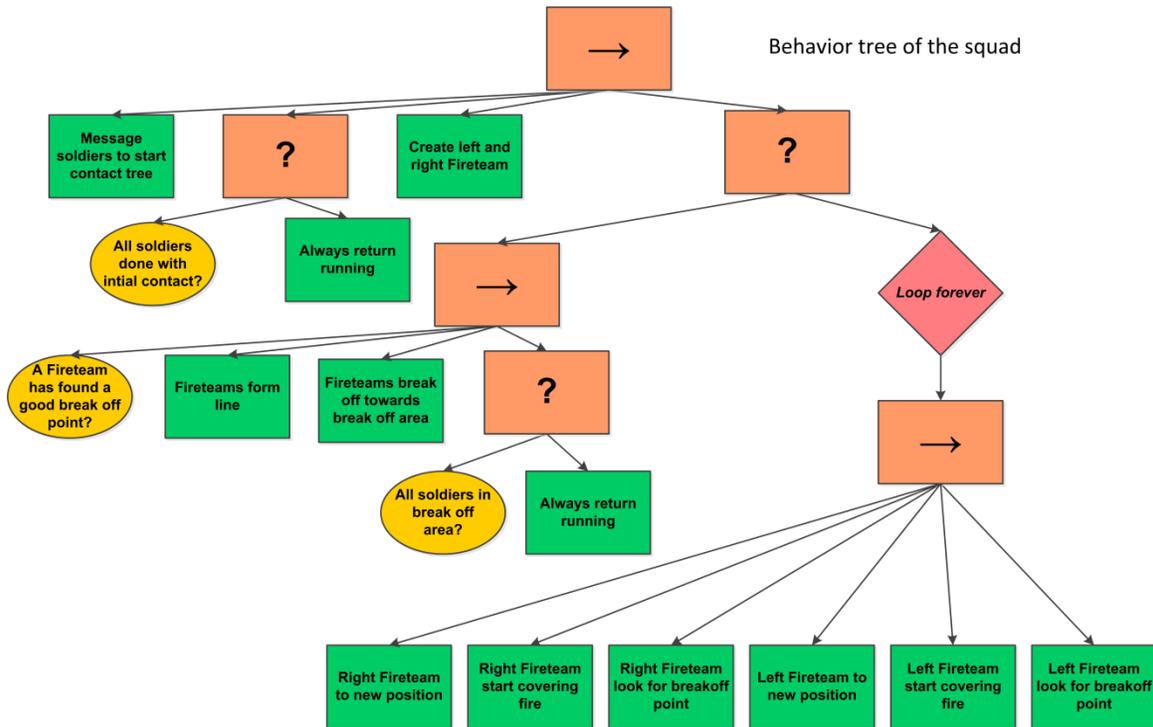


Figure 9. Squad level behavior tree for the contact drill

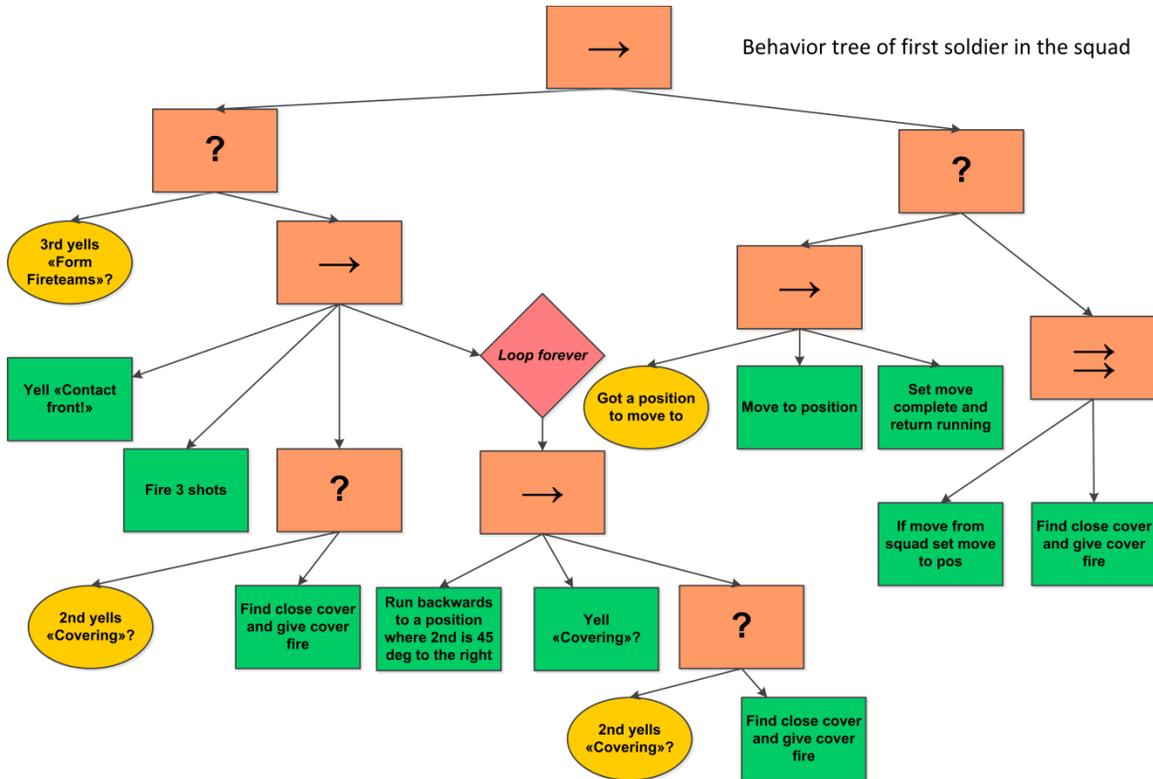


Figure 10. Behavior tree for the contact drill for the first soldier in the squad (Soldier 1)

EXPERIENCES

We have been working with BTs for almost two years. The first year we mostly explored and tested out the approach, but for last year we have systematically developed BTs for battle drills. BTs have a somewhat steeper learning curve than for example FSMs, and it takes some time to become familiar with how the control flow nodes work. Most BT tutorials found on the Internet mostly focus on the basic principles of BTs, and how to implement BTs in an AI engine, but there are few examples on how to create working BTs. We found that hands-on experimentation with different control flow patterns was very useful to better understand how BTs work, and how to create good BTs.

The experiences with using BTs to develop behavior models have so far been very good. Especially the composability and modularity that enables sub-trees to be reused is very useful and simplifies the development process. Often we start by building an initial sub-tree for an action, and then reuse this in multiple places. Later, when we improve the sub-tree, this improvement will apply to all BTs that use this sub-tree.

A good visual editor is very helpful for creating BTs. Visual editors let the user rapidly visualize the flow of the behavior and easier understand how the behavior works. We use VBS Control Editor, which also has a built-in debugger that lets the user add breakpoints and run through a BT step by step while the behavior model is executed in VBS. This functionality is very helpful for pinpointing the source to why the entities sometimes do not behave as intended.

We have found it to be advantageous to have a separate BT handling the internal coordination of a unit, which is not tied to a specific entity (or sub-unit). This way, for example, all the soldiers in a squad can have the same BT, and it is not necessary to have a specific BT for the squad leader. This system also makes it easier to handle casualties, and the unit will still function if the squad leader is killed.

One challenge we have found with using BTs, is handling unexpected jumps in the execution of a BT. An entity can be doing something specific far down in the tree structure, and then suddenly something fundamentally changes about the situation and a very different sub-tree starts executing instead. When this happens, it is important that the entity is reconfigured for the new situation. It is good practice, at the start of a sub-tree, to always make sure things are configured correctly before continuing and not assume anything was correctly configured earlier.

FURTHER WORK

A lot of work remains before the library of behavior models of battle drills is complete. The behavior models also need to be validated, and this will typically be done by face validation by military SMEs and officers from the Norwegian Army. Furthermore, as soon as we start to use the behavior models in our simulations, the behavior models will be continuously subjected to face validation by the players/operators, and in the beginning we expect that the models have to be continuously improved based on feedback from the players.

We have demonstrated the behavior models for officers from the Norwegian Army at several occasions, and the feedback we have received is very positive. The Norwegian Army has expressed interested in the possibility of using the behavior models in other simulation systems that support BTs as well. To make this possible we need to ensure that the models also exist as conceptual models that can be implemented in other simulation systems. The BTs in Figure 9 and Figure 10 are examples such conceptual models. The parts of the models that typically need to be implemented specifically for each simulation system are the task nodes (i.e. the leaf nodes in the BTs).

We are primarily developing behavior models for pure constructive simulations with SAF. This generally sets lower requirements for how high fidelity the behavior models for the entities need to have, compared to behavior models for constructive entities that are meant to be used together with virtual entities directly controlled by humans. This is an issue we may also need to address in future, since it may be desirable to combine virtual and constructive simulations.

SUMMARY AND CONCLUSION

This paper has given an introduction to BTs, discussed advantages and limitations of BTs, and described how we use BTs to build a library of behavior models of the most important battle drills for mechanized infantry platoons. BTs have become very popular, especially for creating behaviors for NPCs in computer games, robots, and autonomous vehicles, mainly because they are composable, modular, and reactive. It is our assessment that BTs also are well suited for developing human behavior models of moderate complexity for semi-automated forces (SAF) in constructive simulations. Finally, it is worth mentioning that the composability and modularity of BT-based behavior models open up opportunities for collaboration on development and sharing of behavior models of battle drills, for example between NATO and partner nations that mostly have similar doctrines.

ACKNOWLEDGEMENTS

The authors would like to thank our IITSEC Birddog Carla Cropper for her helpful comments and suggestions during the process of developing this paper.

REFERENCES

- Abbott, R.G., Basilico, J.D., Glickman, M.R. & Whetzel, J. (2010). Trainable Automated Forces, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (IITSEC) 2010*, Paper No. 10441.
- Buede, D., DeBlois, B., Maxwell, D. & McCarter B. (2013). Filling the Need for Intelligent, Adaptive Non-Player Characters, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (IITSEC) 2013*, Paper No. 13204.
- Colledanchise, M. & Ögren, P. (2018). *Behavior Trees in Robotics and AI: An Introduction*, arXiv preprint, arXiv:1709.00084.
- Colledanchise, M., Almeida, D. & Ögren, P. (2016). *Towards Blended Reactive Planning and Acting using Behavior Trees*, arXiv preprint, arXiv:1611.00230.
- Evensen, P-I. & Bentsen, D.H. (2016). *Simulation of land force operations – a survey of methods and tools*. FFI-report 2015/01579, Kjeller: Norwegian Defence Research Establishment (FFI).
- Evensen, P-I., Selvaag, K., Bentsen, D.H. & Stien H. (2017). Web-Based GUI System for Controlling Entities in Constructive Simulations, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (IITSEC) 2017*, Paper No. 17043.
- Isla, D. (2005). Handling Complexity in the Halo 2 AI, *Proceedings of the Game Developers Conference (GDC) 2005*.
- Jones, R.M. (2011). Introduction to Cognitive Architectures for Modeling and Simulation (Tutorial Slides), *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (IITSEC) 2011*, Tutorial No. 1144.
- Marzinotto, A., Colledanchise, M., Smith, C. & Ögren P. (2014). Towards a Unified Behavior Trees Framework for Robot Control, *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*.
- Merrill, B. (2013). Building Utility Decisions into Your Existing Behavior Tree, in *Game AI Pro: Collected Wisdom of Game AI Professionals*, Rabin, S. (edited by), CRC Press.
- Millington, I. & Funge, J. (2009). *Artificial Intelligence for Games*, 2. Edition, Morgan Kaufmann.
- Norwegian Army. (2014). *Reglement for patruljenærstrid*, Hefte 1 – Grunnlag, Rena: Norwegian Army Land Warfare Centre.
- Papelis, Y. & Madhavan, P. (2010). Modeling Human Behavior, in *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*, Sokolowski, J.A. & Banks, C.M. (edited by), John Wiley & Sons.
- Rabin, S. (edited by) (2013), *Game AI Pro: Collected Wisdom of Game AI Professionals*, CRC Press.
- Robertson, G. & Watson, I. (2015). Building Behavior Trees from Observations in Real-Time Strategy Games, *Proceedings of the 2015 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*.
- Thorpe, J. (2010). Trends in Modeling, Simulation, & Gaming: Personal Observations About the Past Thirty Years and Speculation About the Next Ten, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (IITSEC) 2010*, Paper No. IF1001.