FFI-RAPPORT

18/01651

Modelling Human Behaviour using Behaviour Trees

Per-Idar Evensen
Håvard Stien
Dan Helge Bentsen

Modelling Human Behaviour using Behaviour Trees

Per-Idar Evensen
Håvard Stien
Dan Helge Bentsen

Norwegian Defence Research Establishment (FFI) 27 November 2018

Keywords
Modellering og simulering
Simuleringsmodeller
Kunstig intelligens
Smarte agenter
Menneskelig atferd
Datagenererte styrker

FFI-rapport
18/01651

Prosjektnummer
1401

ISBN
P: 978-82-464-3102-4
E: 978-82-464-3103-1

Approvers
Halvor Ajer, *Research Manager*

The document is electronically approved and therefore has no handwritten signature.

Summary

Behaviour trees (BTs) are a relatively new and increasingly popular approach for developing behaviour models for artificial intelligence (AI) and intelligent agents. The approach has become especially popular for creating behaviours for non-player characters in computer games, robots, and autonomous vehicles. The first high-profile computer game which 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 behaviour of an agent. The control flow nodes are interior nodes (nodes with one or more children) and contain some decision logic for flow control. The task nodes are leaf nodes (nodes without children) and contain conditional tasks which test some property in the simulated environment (or the real world in the case of robots and autonomous vehicles), or action tasks which 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 subtrees which represent more complex actions, and these actions can be composed into higher level behaviours. Task nodes and action subtrees can be reused, and different subtrees can be developed independently of each other.

BT editors with graphical user interfaces enable users without programming skills to create modular behaviour models. Furthermore, to reduce complexity and ensure readability of the graphical model of large BTs, they can be decomposed into smaller subtrees. Examples of AI engines (or AI middleware) for military simulation systems that use BTs are Virtual Battlespace (VBS) Control and MASA Life.

In this report we focus on BTs for human behaviour modelling, and more specifically on using BTs to model battle drills for computer generated forces in military simulation systems. We conduct constructive entity-level simulations of battalion to brigade level operations for experimentation and analysis purposes. To support this work we are developing human behaviour models for semi-automated forces in VBS, using the new AI framework VBS Control.

First, we briefly describe the background for this work. Secondly, we give a short introduction to human behaviour modelling and artificial intelligence. Thirdly, we give an introduction to BTs and look at the process of developing BTs. We also look at the advantages and limitations of BTs and some of the extensions to the BT concept that have been proposed. Finally, we describe how we are using BTs to build a library of behaviour models of battle drills for mechanized infantry platoons, and provide an example of a modelled battle drill.

Sammendrag

Behaviour trees (BT-er) er en relativt ny og stadig mer populær tilnærming for utvikling av oppførselsmodeller for kunstig intelligens (AI) og intelligente agenter. Tilnærmingen har blitt spesielt populær til å utvikle oppførsel for ikke-spillbare karakterer i dataspill, roboter og autonome kjøretøy. Det første høyprofilerte dataspillet som anvendte BT-er var Halo 2 fra Bungie Software, som ble utgitt i 2004.

BT-er representeres som rettede trær med et hierarki av kontrollflytnoder og oppgavenoder som styrer oppførselen til en agent. Kontrollflytnodene er indre noder (noder som har ett eller flere barn) og inneholder beslutningslogikk for flytkontroll. Oppgavenodene er bladnoder (noder uten barn) og inneholder oppgaver for betingelser som tester en egenskap i det simulerte miljøet (eller den virkelige verden når det gjelder roboter og autonome kjøretøy), eller oppgaver for handlinger som endrer tilstanden til simuleringen (eller den virkelige verden) på en eller annen måte.

Det som gjør BT-tilnærming så kraftfull, er komponerbarheten og modulariteten til BT-ene. Oppgavenoder og kontrollflytnoder settes sammen til subtrær som representerer mer komplekse handlinger, og disse handlingene kan videre settes sammen til oppførsel på høyere nivå. Oppgavenoder og subtrær for handlinger kan gjenbrukes, og forskjellige subtrær kan utvikles uavhengig av hverandre.

BT-editorer med grafiske brukergrensesnitt gjør brukere uten programmeringsferdigheter i stand til å utvikle modulære oppførselsmodeller. Dessuten kan den grafiske modellen til et stort BT dekomponeres til mindre subtrær for å redusere kompleksiteten og dermed øke lesbarheten. Eksempler på AI-motorer (eller AI-mellomvare) for militære simuleringssystemer som bruker BT-er er Virtual Battlespace (VBS) Control og MASA Life.

I rapporten fokuserer vi på bruk av BT-er for å modellere menneskelig oppførsel, og særlig på å bruke BT-er til å modellere stridsdriller for datagenererte styrker i militære simuleringssystemer. Vi utfører “constructive”-simuleringer på entitetsnivå av operasjoner på bataljons- til brigadenivå for eksperimenterings- og analyseformål. For å støtte dette arbeidet utvikler vi menneskelige oppførselsmodeller for semiautomatiske styrker i VBS ved hjelp av AI-rammeverket VBS Control.

Først i rapporten beskriver vi kort bakgrunnen for dette arbeidet. Videre gir vi en kort introduksjon til modellering av menneskelig oppførsel og kunstig intelligens. Etter dette gir vi en introduksjon til BT-er og ser på prosessen med å utvikle dem. Vi ser også på fordelene og begrensningene til BT-er, samt på noen av utvidelsene av BT-konseptet som har blitt foreslått. Til slutt beskriver vi hvordan vi bruker BT-er for å bygge opp et bibliotek av oppførselsmodeller for stridsdriller for mekaniserte infanteritropper og gir et eksempel på en modellert stridsdrill.

Contents

1 Introduction

Behaviour trees (BTs) are a relatively new and increasingly popular approach for developing behaviour models for artificial intelligence (AI) and intelligent agents. The approach has become especially popular for creating behaviours for non-player characters (NPCs) in computer games, robots, and autonomous vehicles. The first high-profile computer game which used BTs was Halo 2 from Bungie Software [1], which was released in 2004.

Finite state machines (FSMs) have long been the most dominant technique for creating behaviour models for computer generated forces (CGF) in military simulations [2][3], but BTs are now also starting to become a popular technique for developing behaviour models for automated and semi-automated constructive units in military simulation systems. FSMs are well suited for implementing well-defined doctrinal behaviour of limited complexity [3]. 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 behaviours that are represented [4][5], and as the number of states increases, the number of possible transitions between the states also increases exponentially [6].

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 trees[1] with a hierarchy of *control flow nodes* and *task nodes* that control the behaviour of an agent. The control flow nodes are interior nodes (nodes with one or more children) and contain some decision logic for flow control. The task nodes are leaf nodes (nodes without children) and contain conditional tasks which test some property in the simulated environment (or the real world in the case of robots and autonomous vehicles), or action tasks which alter the state of the simulation (or the real world) in some way [4].

What makes BTs so powerful is their *composability* and *modularity*. Task nodes and control flow nodes are composed into subtrees which represent more complex actions, and these actions can be composed into higher level behaviours [7]. Task nodes and action subtrees can be reused, and different subtrees can be developed independently of each other.

BT editors with graphical user interfaces (GUIs) enable users (e.g. military simulation users) to create modular behaviour 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 subtrees. Examples of AI engines (or AI middleware) for military simulation systems that use BTs are Virtual Battlespace (VBS) Control from Bohemia Interactive Simulations (BISim) and MASA Life from MASA Group.

In this report we focus on BTs for human behaviour modelling, and more specifically on using BTs to model battle drills for computer generated forces (CGF) in military simulation systems.

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

We conduct constructive entity-level simulations of battalion to brigade level operations for experimentation and analysis purposes [4][8][9]. To support this work we are developing human behaviour models for semi-automated forces (SAF) in VBS, using the new AI framework VBS Control.

Since BTs are a quite new technique for developing behaviour models, there is still not very much documentation on how BTs work and how to create BTs. Most BT tutorials found on the Internet focus only on the basic principles of BTs, and how to implement BT functionality in an AI engine, but there are few examples on how to create working BTs. The objective of this report is to help military users create human behaviour models for combat simulations using BTs. The composability and modularity of BT-based behaviour models open up opportunities for collaboration on development and sharing of behaviour models of battle drills, for example between NATO (North Atlantic Treaty Organization) and partner nations, that mostly have similar doctrines. It is our hope that the VBS users from NATO and partner nations over time will be able to jointly create a comprehensive library of modular behaviour models for entities in VBS.

We have been working with BTs for about two years. The first year we mostly explored and tested out the approach, but for the 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. 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.

This report has been organized as follows: Firstly, in Chapter 2, we briefly describe the background for this work. Next, in Chapter 3, we give a short introduction to human behaviour modelling and artificial intelligence. In Chapter 4 we give an introduction to BTs and look at the process of developing BTs. After this, in Chapter 5, we look at the advantages and limitations of BTs, and some of the extensions to the BT concept that have been proposed. Finally, in Chapter 6 we describe how we are using BTs to build a library of behaviour models of battle drills for mechanized infantry platoons, and provide an example of a modelled battle drill.

The work with implementing behaviour models of battle drills for mechanized infantry platoons is mainly being done in FFI-project 1401 “Combat systems – manoeuvre II”.

2 Background

As mentioned in the Introduction, we conduct simulations of land force operations for experimentation and analysis 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 [4][8][9].

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 take advantage of this terrain [4][8][9]. For example, we expect that these two factors will result in more realistic detection and engagement distances in our constructive simulations [8][9]. 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 behaviour models for experimenting with different tactical behaviours of entities and platoons. The composability, modularity, and readability of BTs make this modelling 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 defence 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), which has been named webSAF [8][9]. The entities simulated in VBS will use behaviour models developed in VBS Control. VBS Control is a new framework for BT-based AI in VBS. To get better tactical AI in our simulations, we are currently building a BT-based library of behaviour models for the most important battle drills for mechanized infantry platoons. We will describe the structure of this library more closely in Chapter 6.2.

3 Human behaviour modelling and artificial intelligence

Human behaviour is the "collective set of actions exhibited by human beings, either individually or in groups of various sizes and compositions" [5]. Factors that may determine and affect human behaviour include physical properties (e.g. strength, endurance), cognitive properties (e.g. memory, reasoning), and social properties (e.g. cultural norms, role in social group) [10].

Modelling realistic human behaviour and cognition, including decision-making and creativity, is the hardest and most complex challenge in combat simulation [11]. Human behaviour modelling is challenging because "[h]uman behaviour is not generally yet thought to obey observable laws" [12]. "In general, the behaviour of large number of human beings does not currently appear to behave in accordance with deterministic rules" [12]. Consequently, the current status for human behaviour simulation is that it can be used "to understand, [but] not necessarily predict, the aggregate behavior of an inherently complex system for which we have no better model" [5]. When using human behaviour models "it is often possible to perform sensitivity analysis and identify broad trends as opposed to exact predictions" [5]. For example, a simulation using CGF may show that increasing the number of main battle tanks (MBTs) has a positive effect on the outcome of a scenario, but it cannot be used to pinpoint the exact number of MBTs required to win the battle with a certain probability [5].

Human behaviour can be divided into the physical, tactical, and strategic level, based on the complexity of the goal of the behaviour and the duration of the performed activity [5]. At the physical level human behaviour is driven by physiology and automated processes like stimulus response and motor skills. "Decisions are done at an instinctive or reactive level, and emotions have little impact on the process; instead, performance is governed by the level of workload, fatigue, situational awareness, and other similar factors" [5]. Examples of this level of behaviour are walking, driving a vehicle, and firing a weapon. Human behaviour at the tactical level is driven by short-term goals and includes tactical decision-making and emotions [5]. At the strategic level human behaviour involves long-term planning and complex, high-level decision-making based on experience, intuition, and emotions [5].

Human behaviour at the physical level can be modelled using physics-based models and performance data. "[T]actical and strategic behaviors are harder to model due to the adaptive and unpredictable nature of human behavior. When incorporating larger populations, the complexity drastically increases to the point where such models are difficult, if not impossible, to validate" [5].

There are mainly two schools of thought for modelling the higher levels of human behaviour. The first considers human beings as rational entities, and focuses on modelling rational decision-making to achieve a specific goal based on deterministic or stochastic approaches, ignoring the effect of emotions. The second considers human beings as quasi-rational entities that still pursue a specific goal, but frequently make suboptimal decisions and even exhibit actions that can act

contrary to their goal [5]. “Rational decision making is by far easier to model and simulate than quasi-rational behavior” [5]. The processes that govern suboptimal decision-making are complex, and are not yet fully understood [5].

3.1 Artificial intelligence

Artificial intelligence (AI) is the field of study for creating intelligence exhibited by machines or software. An autonomous intelligent entity is referred to as an *intelligent agent* (or an autonomous intelligent agent). An intelligent agent observes the environment through sensors and acts upon the environment using actuators in order to pursue its goals. It may also learn or use knowledge to achieve its goals. An agent can also be an aggregated unit. A *multi-agent system* (MAS) is a system composed of multiple interacting intelligent agents within an environment. Figure 3.1 shows a proposed architecture for generic intelligent agents [13][14]. Agent architectures that more specially attempt to model human cognition are referred to as *cognitive architectures*. It should be noted that, whereas AI systems in general are designed to complete tasks faster and with fewer errors than human beings, human behaviour models are designed to complete tasks in the same way as humans are expected to complete the tasks [15].

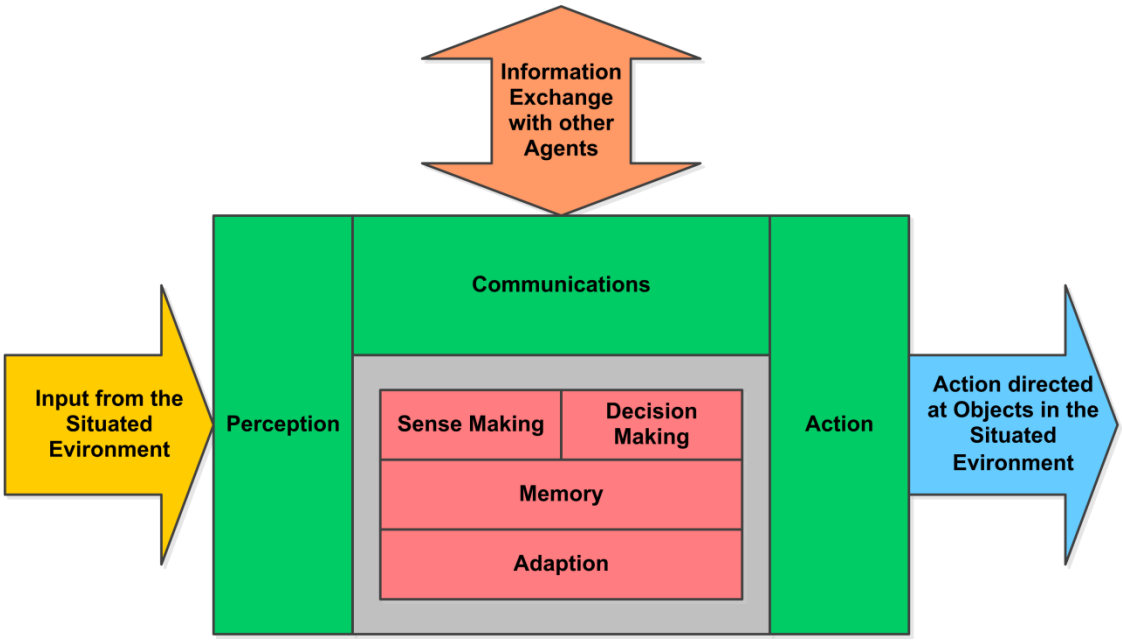


Figure 3.1 Architecture for generic intelligent agents ([13][14]).

4 Behaviour trees

In this chapter we will look at how BTs are structured and how they work. BTs are a relatively new technique for developing behaviour models. Nevertheless, BTs have reached a certain maturity and have been treated in at least two Game AI textbooks [7][16]. Furthermore, a number of BT tutorials can be found on the Internet. The available descriptions of BTs differ slightly, and a suggested unified framework for BTs was published in 2014 [17].

4.1 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 has 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. Figure 4.1 shows an example of a directed rooted tree.

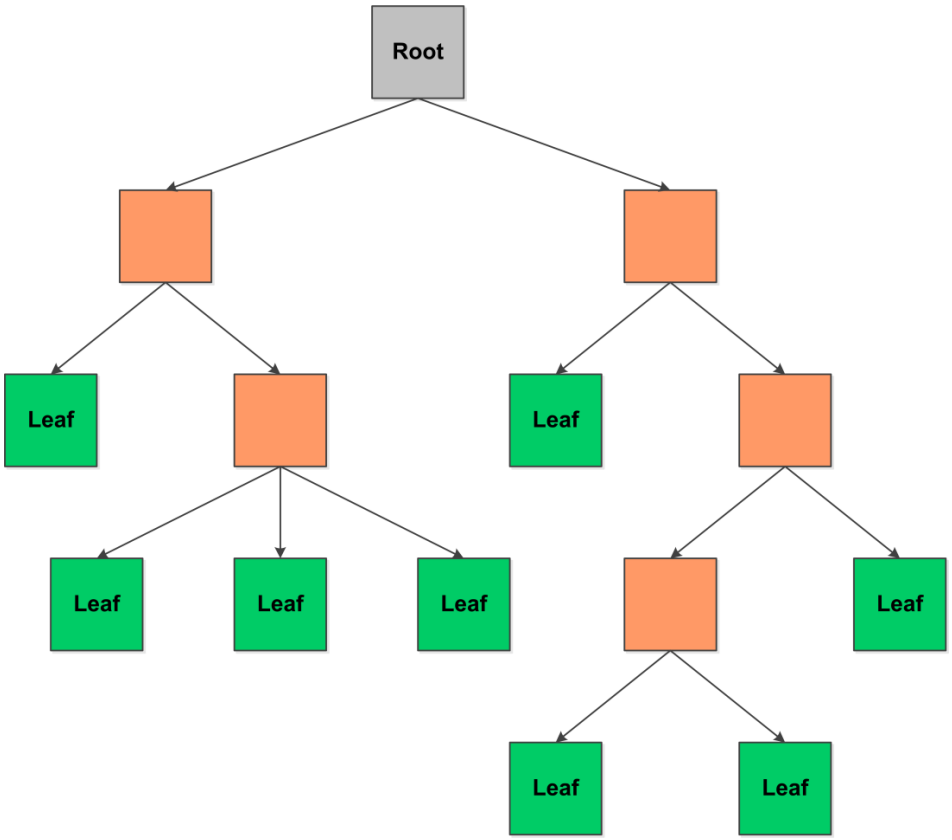


Figure 4.1 Example of a directed rooted tree.

4.2 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 [7]. 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. Figure 4.2 shows the order in which the nodes are visited in a depth-first traversal of the directed rooted tree from Figure 4.1.

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 can also be longer. The simulation step represents the maximum time it will take for the behaviour model to detect, and be able to react to, a change in the environment. For a model representing human behaviour this should not be longer than a typical human reaction time (usually between 0.2 and 0.4 seconds).

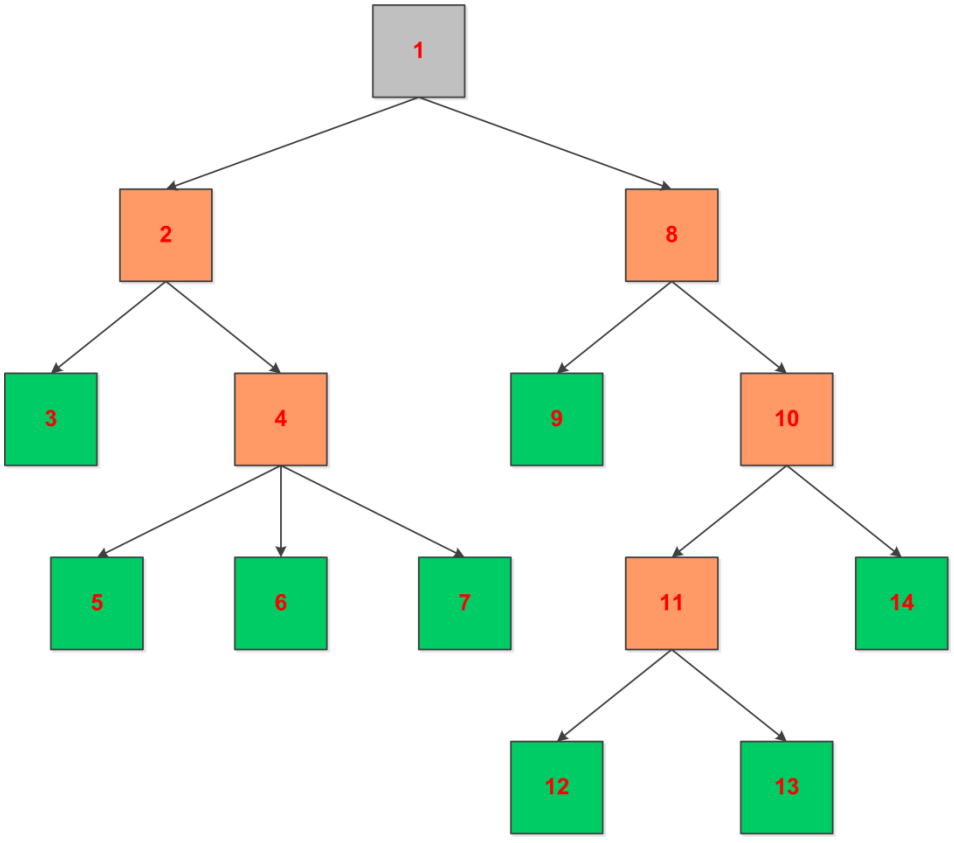


Figure 4.2 The order in which the nodes are visited in depth-first traversal of a directed rooted tree.

During the traversal of a BT each executed 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, for example 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 which 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 more commonly, to restart the BT execution at the root. Starting at the root at each simulation step allows the behaviour 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.

4.3 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 subtree.

4.3.1 Control flow nodes

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*.

4.3.1.1 Selector nodes

A selector[2] 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 4.3 shows the graphical representation of a selector node with *N* children. The selector node is denoted by a question mark

[2] Selector nodes are sometimes called fallback nodes.

(?). Algorithm 4.1 shows the pseudo code for the execution of a selector node with *N* children. The *Tick()* function is the periodic execution signal that propagates through the BT.

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.

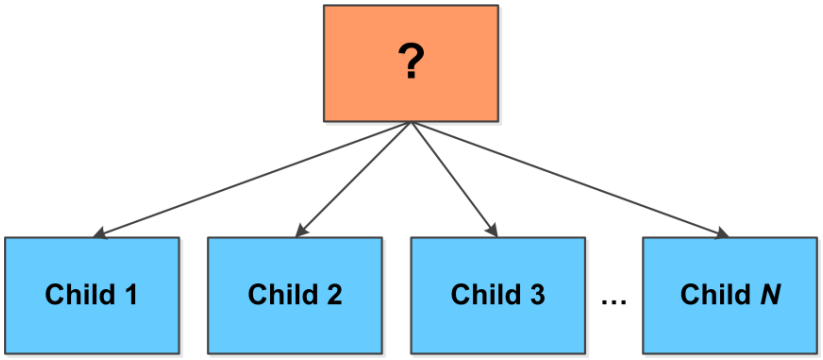


Figure 4.3 Graphical representation of a selector node with N children.

```
for (i = 1; i <= N; i++) {
    childStatus = Tick(child(i));
    if (childStatus == running) {
        return running;
    }
    else if (childStatus == success) {
        return success;
    }
}
return failure;
```

Algorithm 4.1 Pseudo code for the execution of a selector node with N children.

4.3.1.2 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 4.4 shows the graphical representation of a sequence node with *N* children. The sequence node is denoted by a rightwards

arrow (→). Algorithm 4.2 shows the pseudo code for the execution of a sequence node with *N* children.

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.

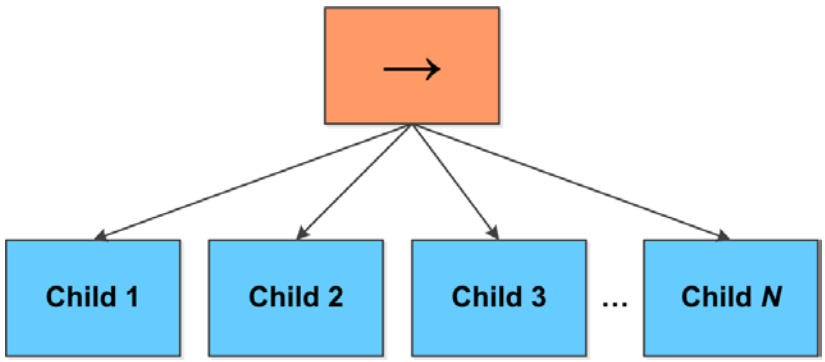


Figure 4.4 Graphical representation of a sequence node with N children.

```
for (i = 1; i <= N; i++) {
    childStatus = Tick(child(i));
    if (childStatus == running) {
        return running;
    }
    else if (childStatus == failure) {
        return failure;
    }
}
return success;
```

Algorithm 4.2 Pseudo code for the execution of a sequence node with N children.

4.3.1.3 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 4.5 shows the graphical representation of a parallel node with *N*

children. The parallel node is denoted by rightwards paired arrows (⇉). Algorithm 4.3 shows the pseudo code for the execution of a parallel node with *N* children. Logically the children are executed in parallel. However, in an implementation their *Tick()* functions are typically called sequentially within the same tick without waiting for the call to return before moving to the next child.

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.

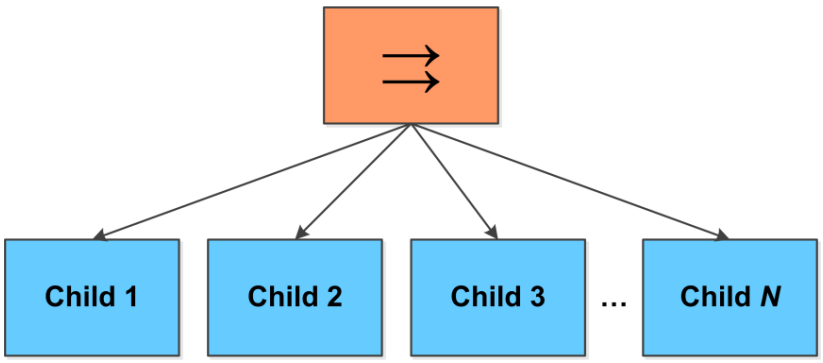


Figure 4.5 Graphical representation of the parallel node.

```
for (i = 1; i <= N; i++) {
    Tick(child(i));
}
if (nChildStatusSuccess == N) {
    return success;
}
else if (nChildStatusFailure >= 1 {
    TerminateAllRunningChildren();
    return failure;
}
else {
    return running;
}
```

Algorithm 4.3 Pseudo code for the execution of a parallel node with N children.

4.3.2 Task nodes

The task nodes are the leaf nodes in a BT. There are two types of task nodes: *condition nodes* and *action nodes*.

4.3.2.1 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 4.6 (to the left) shows the graphical representation of the condition node. The "*Condition*" text label will typically be a description of the condition. Algorithm 4.4 shows the pseudo code for the execution of a condition node.

Figure 4.6 Graphical representation of the condition node (to the left) and the action node (to the right). The "Condition" and "Action" text labels will typically be descriptions of the condition and the action respectively.

```
if (condition == true) {
    return success;
}
else {
    return failure;
}
```

Algorithm 4.4 Pseudo code for the execution of a condition node.

4.3.2.2 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 4.6 (to the right) shows the graphical representation of the action node. The "*Action*" text label will typically be a description of the action. Algorithm 4.5 shows the pseudo code for the execution of an action node. The *Tick()* function will execute the action until it finishes, but no longer than the end of the simulation step.

```
Tick(Action);
if (Action completed) {
    return success;
}
else if (Action failed) {
    return failure;
}
else {
    return running;
}
```

Algorithm 4.5 Pseudo code for the execution of an action node.

4.3.3 Decorator nodes

A decorator node is a special type of node that has only one child and modifies the behaviour 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), *succeeder* 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 4.7 shows the graphical representation of the decorator node. The "*Decorator*" text label will typically be a description of the rule of the decorator.

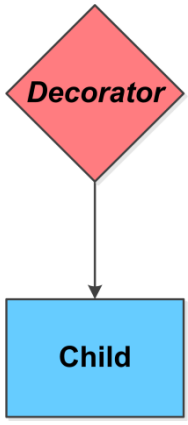


Figure 4.7 Graphical representation of the decorator node and its child. The "Decorator" text label will typically be a description of the rule of the decorator.

4.3.4 Reference nodes

A reference node is simply a proxy node that refers to a particular subtree. A particular subtree may be used several places in a BT.

4.3.5 Summary of node types

Table 4.1 summarizes the properties of the different types of nodes in a BT.

Node type	Symbol	Success	Failure	Running
Selector	?	If one child returns success	If all children return failure	If one child returns running
Sequence	→	If all children return success	If one child returns failure	If one child returns running
Parallel	⇉	If all children return success	If one child returns failure	If one or more children return running (and none of the children have returned failure)
Action	*Text label*	If action was completed	If action could not be completed	If action is still running
Condition	*Text label*	If condition is fulfilled	If condition is not fulfilled	Never
Decorator	*Text label*	According to rule	According to rule	According to rule

Table 4.1 Properties of the different types of nodes in a BT.

4.4 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 subtree can be placed at any position in the 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.

4.5 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[3], the average

[3] *O* notation (O stands for order) classifies algorithms according to how their running time or memory requirements grows as the number of elements processed by the algorithm grows.

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 [7].

4.6 Developing behaviour 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 behaviours, while the right branch of the tree will contain the low-priority behaviours. The default or unconditional behaviour 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 [18]. Generally, most modularity is achieved if each task can be broken into the smallest parts that can usefully be composed [7]. However, a BT that is too fine grained may be unnecessarily complex [18]. 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 behaviour by using machine learning techniques [19][20].

4.7 Examples

4.7.1 Example 1: Move into room (simple)

Figure 4.8 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 node, 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 fails, the whole BT will return failure.

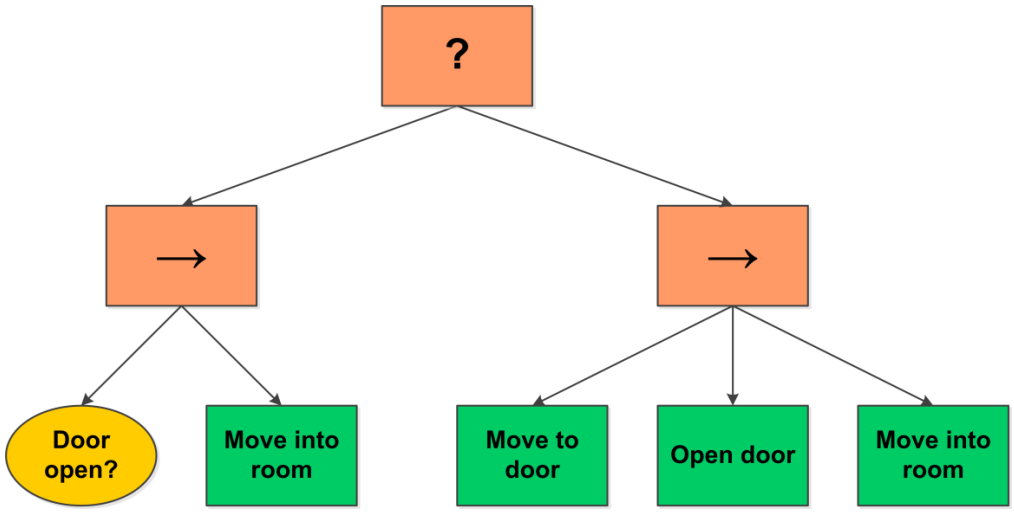


Figure 4.8 Simple BT for moving into a room.

4.7.2 Example 2: Move into room (more advanced)

Figure 4.9 shows a slightly more advanced BT for moving into a room. Here the “Open door” action node from Figure 4.8 has been extended to a subtree starting with selector node. This selector node will first try to execute the left child, which is a sequence node. The sequence node will first execute the condition node, which checks if the door is locked. If the door is locked, the sequence node will execute the next child, which is an action node that unlocks the door. If the sequence node fails, the selector node will try to execute the right child, which is also a sequence node. This sequence node will try to execute the action node “Kick door”, and then execute a condition node, which checks if the door is now open.

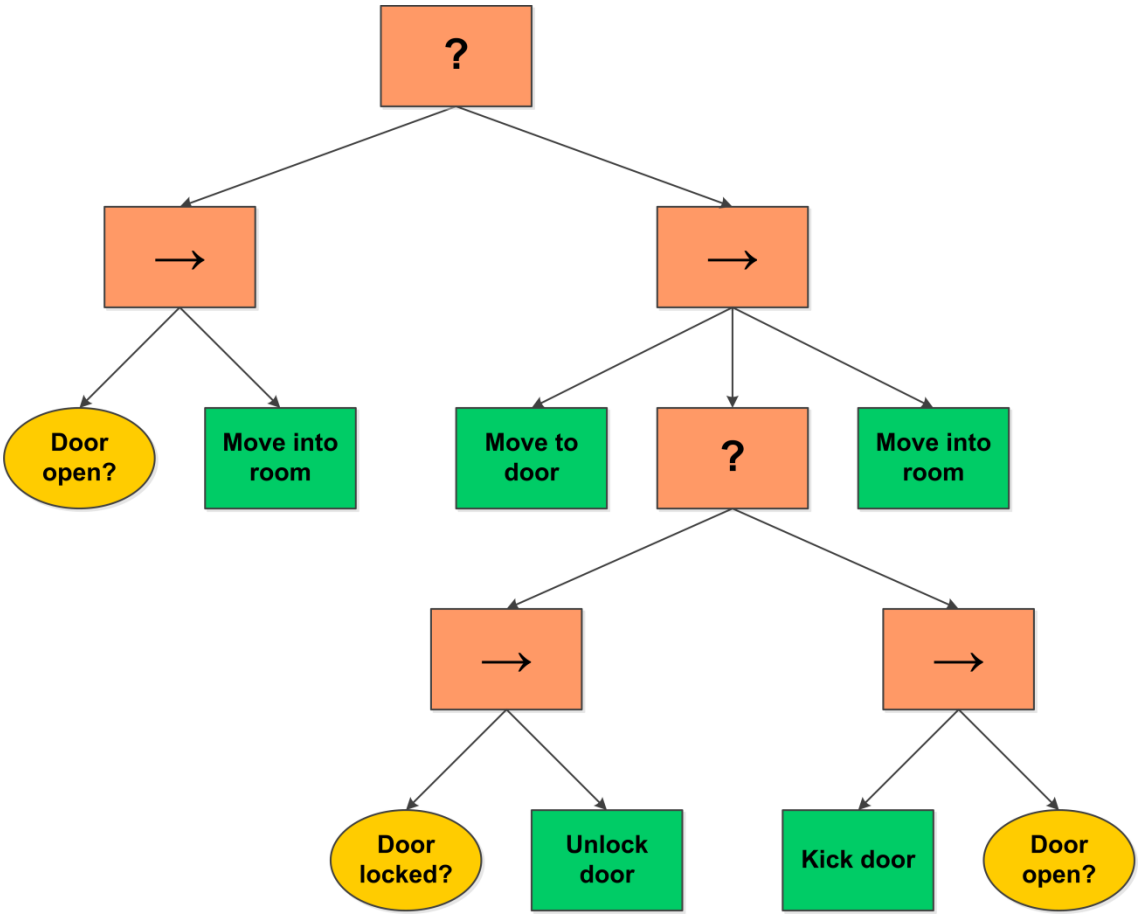


Figure 4.9 More advanced BT for moving into a room.

5 Advantages, limitations and possible extensions of behaviour trees

In AI research BTs are classified as a simple form of planning referred to as *reactive planning* [7]. To be reactive means to be able to quickly react to changes. Behaviour 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 behaviour that can be modelled [17]. In practice however, whereas FSMs tend to become impractical and unmanageable as the complexity of the behaviour model grows, complex BTs are much easier to maintain and extend.

In this chapter we will first look at the advantages and limitations of BTs, and then look at some of the extensions to the BT concept that have been proposed.

5.1 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 subtrees 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 behaviour models of moderate complexity for semi-automated forces (SAF) in constructive simulations.

5.2 Limitations

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

- BTs are poor at modelling the uncertainty in situations where there are multiple valid options to choose from [16].
- It is somewhat cumbersome to represent typical state-based behaviour using BTs [7].
- There are limitations on how large and complex behaviour 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.

5.3 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 behaviour. A more comprehensive way to address this limitation is to combine BTs with utility-based decision-making, as suggested in [21]. 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 [7]. Instead of having one large BT representing all the possible behaviours 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 [18]. This approach will, however, make the BT less reactive, and will therefore not be suitable for use in dynamic and unpredictable environments.

6 Modelling battle drills for computer-generated forces using behaviour trees

In this chapter we first give a short description of VBS Control. Then we describe how we are using BTs to build a library of behaviour models of battle drills for mechanized infantry platoons, and provide an example of a modelled battle drill.

6.1 VBS Control

VBS Control is BISim's new framework for AI in VBS. The behaviour models used by VBS Control are based on BTs, and users can create customized behaviour models through the VBS Control Editor. BTs can be visually debugged in real time, and VBS can visualize the path planning and navigation mesh for debugging purposes. Figure 6.1 shows a BT in VBS Control Editor (to the left) and a visualization of the path planning and navigation mesh in VBS (to the right).

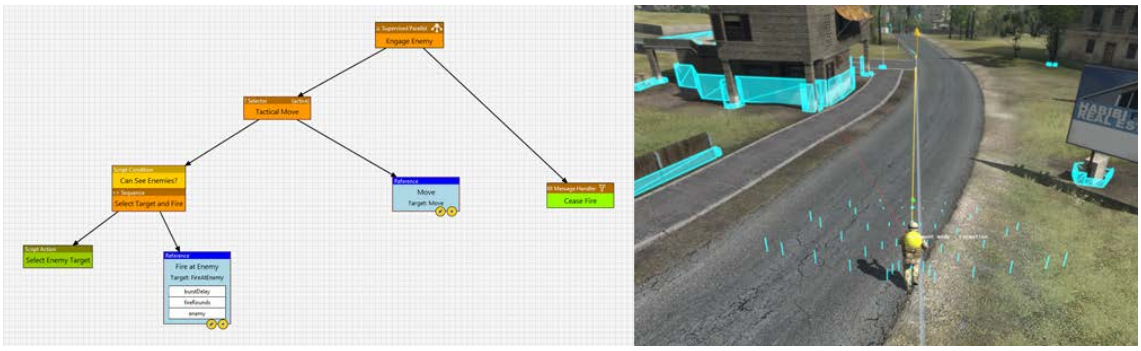


Figure 6.1 A behaviour tree in VBS Control Editor (to the left) and a visualization of the path planning and navigation mesh in VBS (to the right) (Bohemia Interactive Simulations).

6.2 Building a library of behaviour models of battle drills

As mentioned in Chapter 2, to get better tactical AI in our simulations, we are currently building a BT-based library of behaviour 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 behaviour 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 behaviour 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 manoeuvre forces.

The workflow for creating the behaviour 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 subtrees 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 subtrees 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 behaviour models coordinate the behaviour of the individual entities and issue orders (i.e. assigns BTs) to the entities. The unit level behaviour 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 behaviour models. For example for infantry, we have behaviour 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 modelled in VBS Control, and in the future we also plan to build BTs at the company level. Figure 6.2 illustrates the hierarchy of behaviour models.

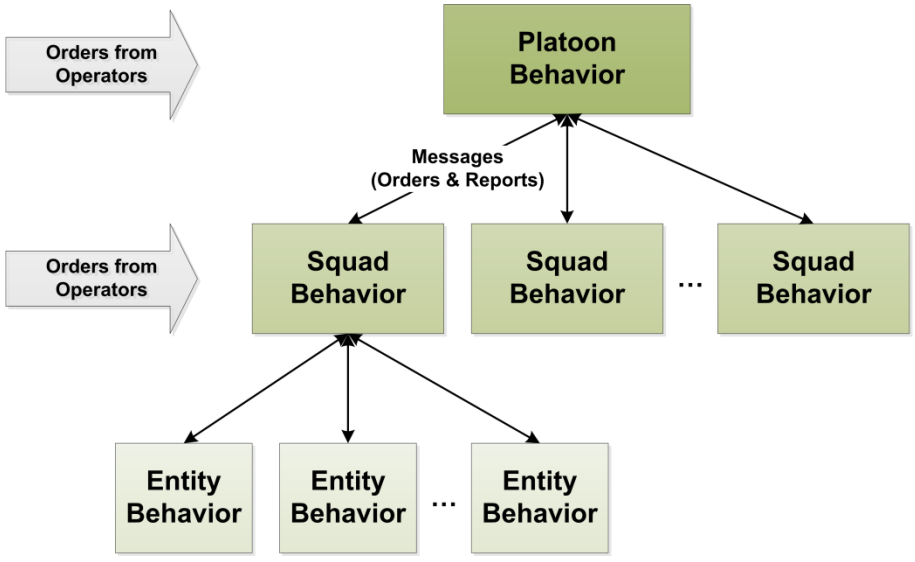


Figure 6.2 Hierarchy of behaviour models.

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 of course 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 6.3 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.

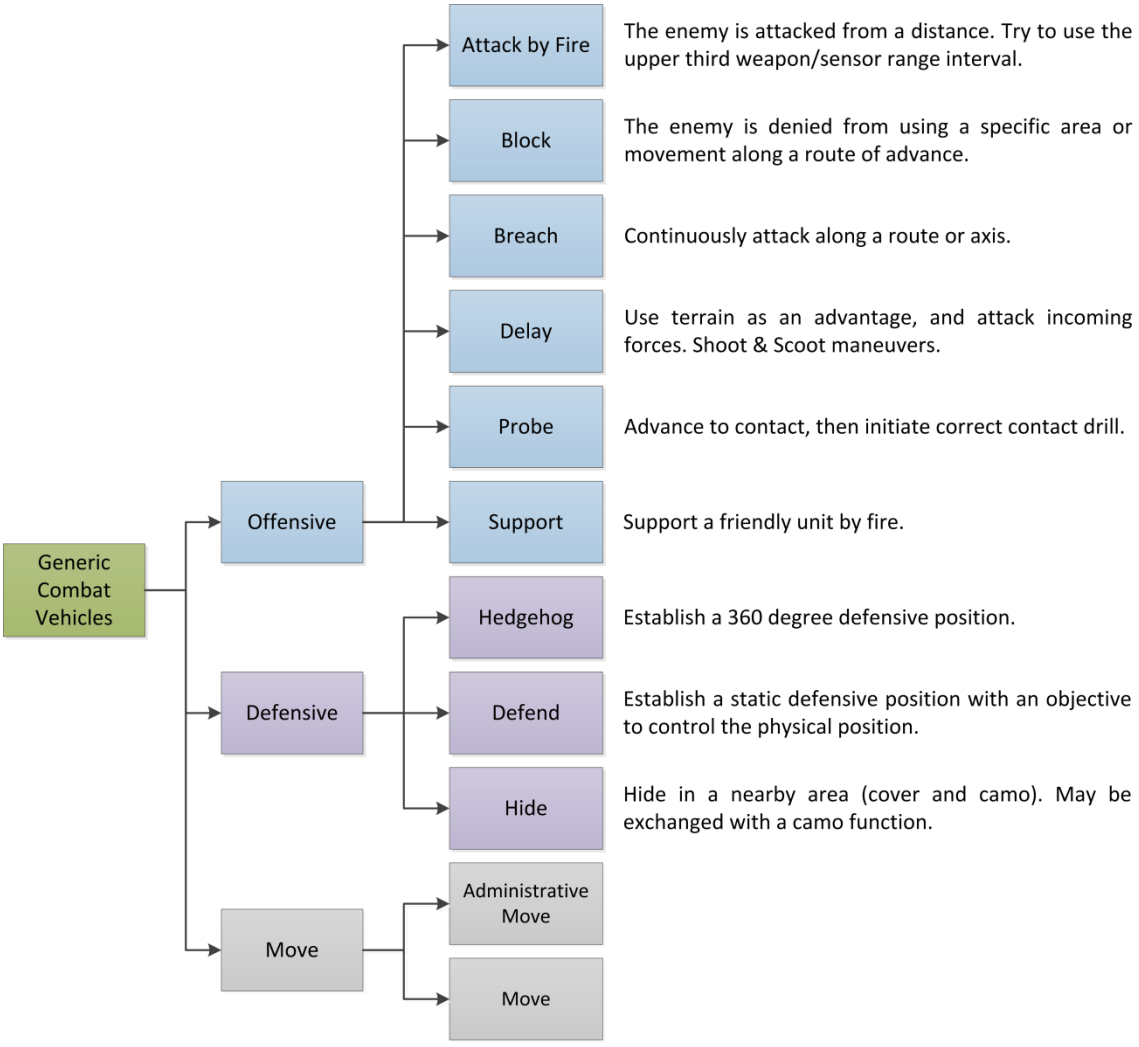


Figure 6.3 Example of orders for combat vehicle platoons.

6.3 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 [22]. After this we will look at the modelled squad level and entity level BTs for the contact drill.

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 6.4 (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 6.4 (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 6.4 (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 6.4 (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 6.4 (e).

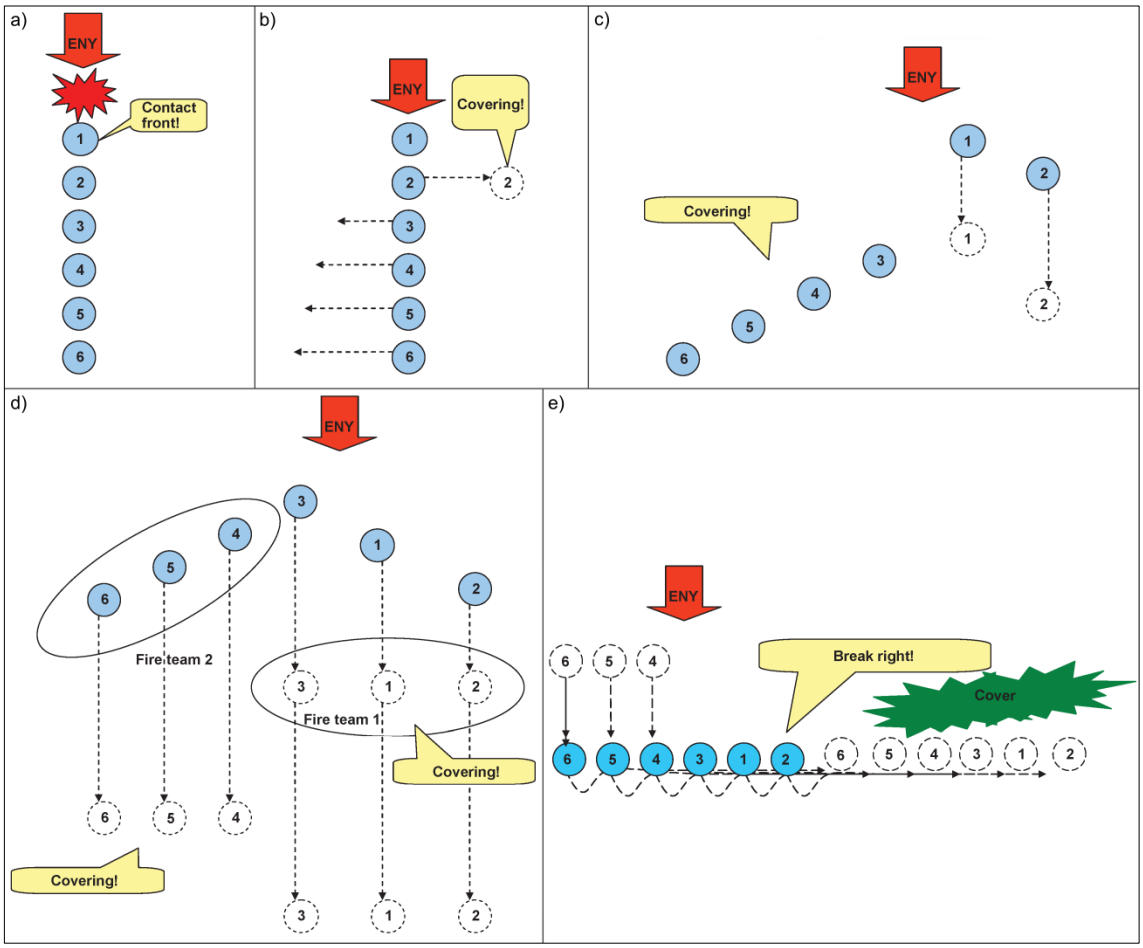


Figure 6.4 Battle drill for enemy contact for a dismounted infantry squad ([22]).

6.3.1 Modelled behaviour trees for contact drill

Figure 6.5 shows the modelled BT for the squad, and Figure 6.6 shows the modelled BT for the first soldier in the squad (Soldier 1). The squad BT for the contact drill will typically be used as a subtree in a BT for a squad move order.

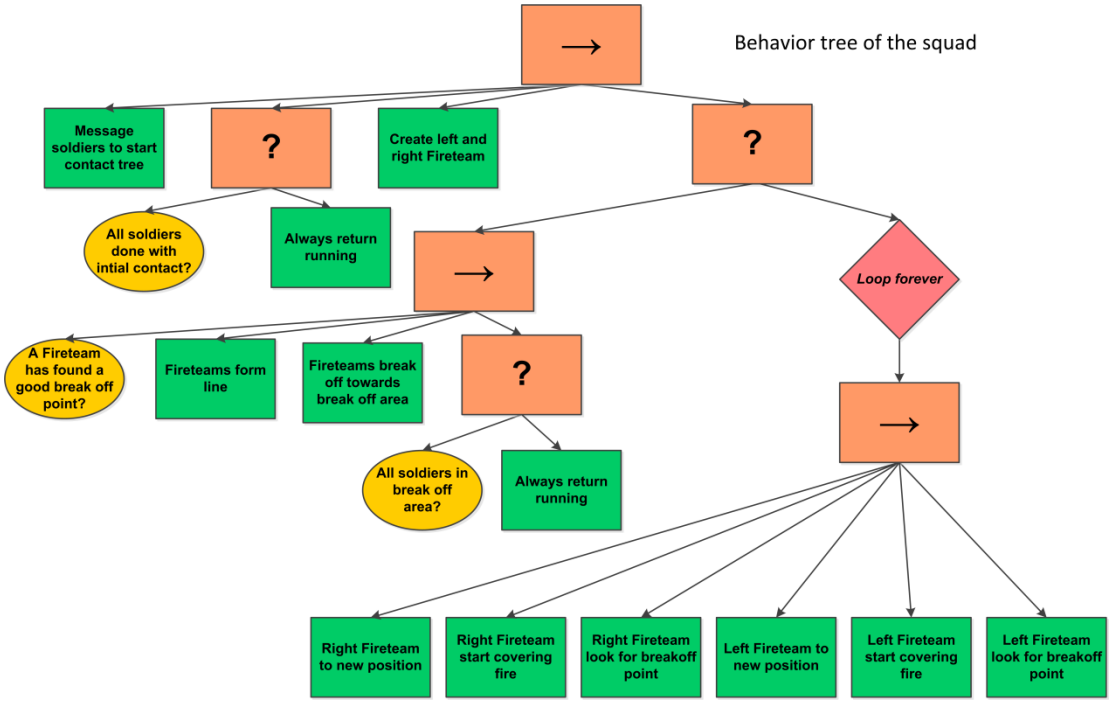


Figure 6.5 Squad level behaviour tree for the contact drill.

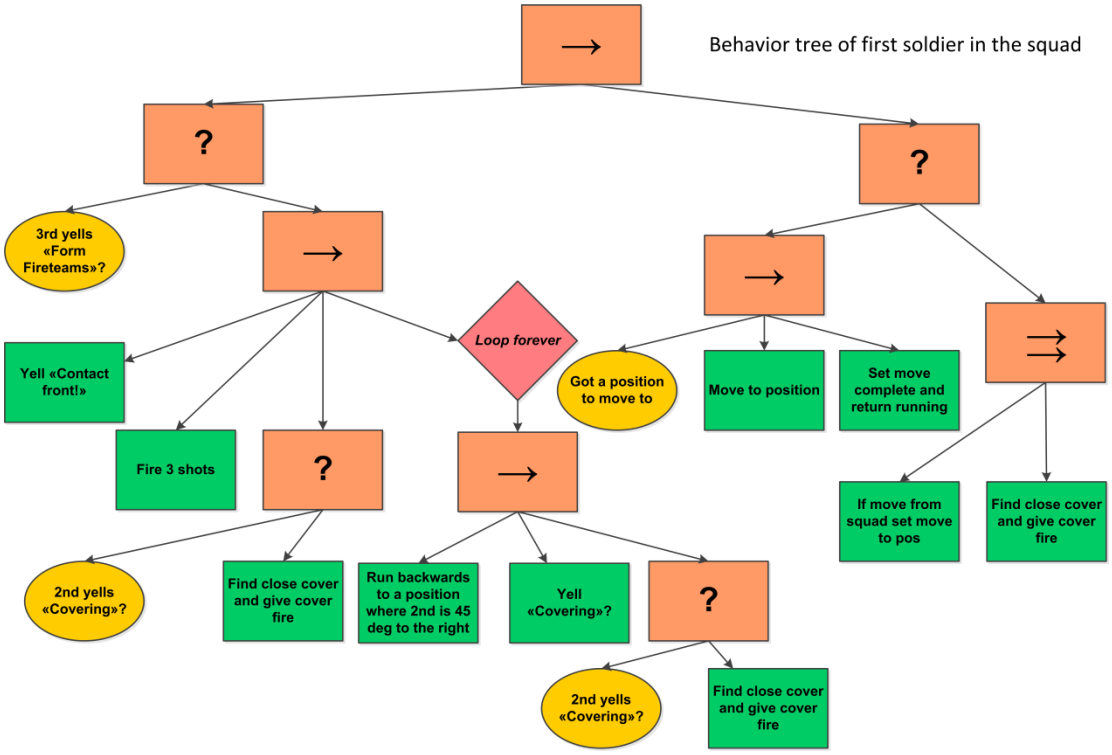


Figure 6.6 Behaviour tree for the contact drill for the first soldier in the squad (Soldier 1).

6.4 Further work

A lot of work remains before the library of behaviour models of battle drills is complete. The behaviour 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 behaviour models in our simulations, they 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 behaviour models for officers from the Norwegian Army at several occasions, and the feedback we have received is very positive. The Norwegian Army has expressed interest in the possibility of using the behaviour 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 6.5 and Figure 6.6 are examples of 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 behaviour models for pure constructive simulations with SAF. This generally sets lower requirements for how high fidelity the behaviour models for the entities need to have, compared to behaviour 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 the future, since it may be desirable for us to combine virtual and constructive simulations.

7 Summary and conclusion

This report has given an introduction to *behaviour trees* (BTs), discussed advantages and limitations of BTs, and described how we use BTs to build a library of behaviour models of the most important battle drills for mechanized infantry platoons. BTs have become very popular, especially for creating behaviours for NPCs in computer games, robots, and autonomous vehicles, mainly because they are *composable*, *modular*, and *reactive*. These properties also make BTs well suited for developing human behaviour models of moderate complexity for semi-automated forces (SAF) in constructive simulations.

The experiences with using BTs to develop behaviour models have so far been very good. However, a lot of work still remains before our library of behaviour models of battle drills is complete.

Finally, the composability and modularity of BT-based behaviour models open up opportunities for collaboration on development and sharing of behaviour models of battle drills, for example between NATO and partner nations that mostly have similar doctrines.

References

[1] D. Isla, Handling Complexity in the Halo 2 AI, *Proceedings of the Game Developers Conference (GDC) 2005*, 2005.

[2] R.M. Jones, Introduction to Cognitive Architectures for Modeling and Simulation (Tutorial Slides), *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2011*, Tutorial No. 1144, 2011.

[3] R.G. Abbott, J.D. Basilico, M.R. Glickman & J. Whetzel, Trainable Automated Forces, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2010*, Paper No. 10441, 2010.

[4] P-I. Evensen & D.H. Bentsen, *Simulation of Land Force Operations – A Survey of Methods and Tool*, FFI-rapport 2015/01579, 2016.

[5] Y. Papelis & P. Madhavan, Modeling Human Behavior, in *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*, J.A. Sokolowski & C.M. Banks (edited by), John Wiley & Sons, 2010.

[6] D. Buede, B. DeBlois, D. Maxwell & B. McCarter, Filling the Need for Intelligent, Adaptive Non-Player Characters, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2013*, Paper No. 13204, 2013.

[7] I. Millington & J. Funge, *Artificial Intelligence for Games*, 2. Edition, Morgan Kaufmann, 2009.

[8] P-I. Evensen, K. Selvaag, D.H. Bentsen & H. Stien, Web-Based GUI System for Controlling Entities in Constructive Simulations, *Proceedings of the Interservice/ Industry Training, Simulation and Education Conference (I/ITSEC) 2017*, Paper No. 17043, 2017.

[9] P-I. Evensen, K. Selvaag, D.H. Bentsen, H.R. Holhjem & H. Stien, Easy-to-Use, Web-Based Graphical User Interface for Controlling Entities in Constructive Simulations, *Proceedings of the NATO Modelling and Simulation Group (NMSG) Annual Symposium 2018 (STO-MP-MSG-159)*, Paper No. 7, 2018.

[10] G. Gunzelmann, C. Gaughan, T. Alexander, W. Huiskamp, K. van den Bosch, S. de Jong, A.G. Bruzzone & A. Tremori, In Search of Interoperability Standards for Human Behaviour Representations, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2014*, Paper No. 14027, 2014.

[11] J. Thorpe, 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 (I/ITSEC) 2010*, Paper No. IF1001, 2010.

[12] J. Storr, *The Human Face of War*, Continuum, 2009.

[13] L.J. Moya & A. Tolk, Towards a Taxonomy of Agents and Multi-Agent Systems, *2007 Proceedings of the Spring Simulation Multi-Conference (SpringSim)*, 2007.

[14] G.K. Bharathy, L. Yilmaz & A. Tolk, Agent Directed Simulation for Combat Modeling and Distributed Simulation, in *Engineering Principles of Combat Modeling and Distributed Simulation*, A. Tolk (edited by), John Wiley & Sons, 2012.

[15] M.J. Doyle & A.M. Portrey, Are Current Modeling Architectures Viable for Rapid Human Behavior Modeling?, *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2011*, Paper No. 11129, 2011.

[16] S. Rabin (edited by), *Game AI Pro: Collected Wisdom of Game AI Professionals*, CRC Press, 2013.

[17] A. Marzinotto, M. Colledanchise, C. Smith & P. Ögren, Towards a Unified Behavior Trees Framework for Robot Control, *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[18] M. Colledanchise & P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*, arXiv preprint, arXiv:1709.00084, 2018.

[19] M. Colledanchise, D. Almeida & P. Ögren, *Towards Blended Reactive Planning and Acting using Behavior Trees*, arXiv preprint, arXiv:1611.00230, 2016.

[20] G. Robertson & I. Watson, 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)*, 2015.

[21] B. Merrill, Building Utility Decisions into Your Existing Behavior Tree, in *Game AI Pro: Collected Wisdom of Game AI Professionals*, S. Rabin (edited by), CRC Press, 2013.

[22] Norwegian Army, *Reglement for patruljenærstrid*, Hefte 1 – Grunnlag, Norwegian Army Land Warfare Centre, 2014.

Abbreviations

AI	Artificial Intelligence
BISim	Bohemia Interactive Simulations
BT	Behaviour Tree
CGF	Computer Generated Forces
DAG	Directed Acyclic Graph
FSM	Finite State Machine
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IFV	Infantry Fighting Vehicle
MAS	Multi-Agent System
MBT	Main Battle Tank
NATO	North Atlantic Treaty Organization
NPC	Non-Player Character
SAF	Semi-Automated Forces
SME	Subject Matter Expert
UTL	Universal Task List
VBS	Virtual Battlespace

About FFI

The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's MISSION

FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's VISION

FFI turns knowledge and ideas into an efficient defence.

FFI's CHARACTERISTICS

Creative, daring, broad-minded and responsible.

Om FFI

Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan med særskilte fullmakter underlagt Forsvarsdepartementet.

FFIs FORMÅL

Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

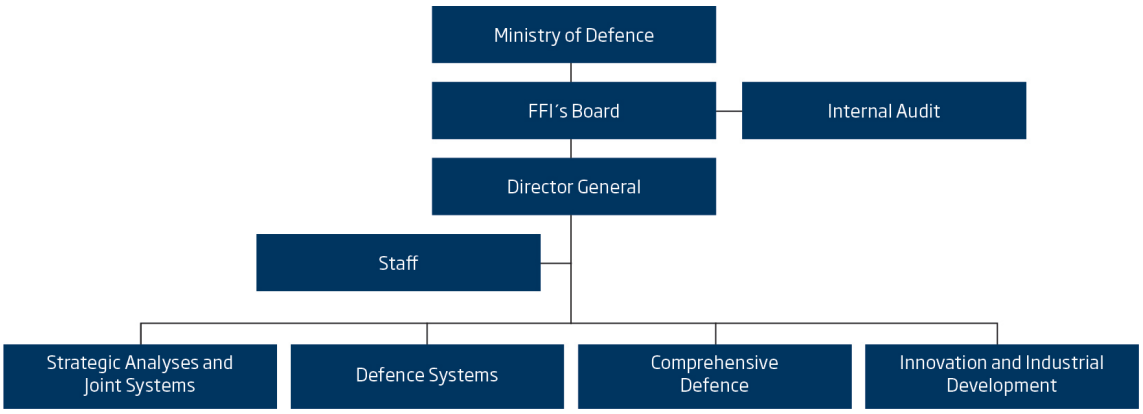
FFIs VISJON

FFI gjør kunnskap og ideer til et effektivt forsvar.

FFIs VERDIER

Skapende, drivende, vidsynt og ansvarlig.

FFI's organisation



Forsvarets forskningsinstitutt
Postboks 25
2027 Kjeller

Besøksadresse:
Instituttveien 20
2007 Kjeller

Telefon: 63 80 70 00
Telefaks: 63 80 71 15
Epost: ffi@ffi.no

Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller

Office address:
Instituttveien 20
N-2007 Kjeller

Telephone: +47 63 80 70 00
Telefax: +47 63 80 71 15
Email: ffi@ffi.no

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment