



FFI Forsvarets
forskningsinstitutt
Norwegian Defence Research Establishment

Using Behaviour Trees to Model Battle Drills for Computer-Generated Forces

Per-Idar Evensen, Håvard Stien & Dan Helge Bentsen
Norwegian Defence Research Establishment (FFI)

NATO Modelling & Simulation Group Annual Symposium (NMSG-171)
Vienna, 24–25 October 2019



Agenda

- Background
- Behaviour trees (BTs)
 - Introduction
 - Developing BTs
 - Advantages and Limitations
- Modelling Battle Drills
 - Example
 - Video
- Experiences with BTs
- Summary and Conclusion



Background

- Research question: *how to increase combat effectiveness in land force operations?*
- Detailed simulations of battalion to brigade level operations, to assess and compare the performance of different land force structures, that may vary with regard to:
 - Composition of *material* and *equipment*
 - *Tactical organization*
 - *Operational concept*
- Two main factors that have the potential to improve the fidelity of our constructive simulations:
 - Increased *terrain resolution*
 - Better *tactical artificial intelligence* (AI) that can take advantage of this terrain



Behaviour Trees

- ***Behaviour trees*** (BTs) are a relatively new and increasingly popular approach for developing behaviours for artificial intelligence (AI) and intelligent agents:
 - Non-player characters (NPCs) in computer games, robots, and autonomous vehicles
 - First high-profile computer game which used BTs was Halo 2 from Bungie Software (released in 2004)
- What makes BTs so powerful is their ***composability*** and ***modularity***:
 - Task nodes and control flow nodes are composed into sub-trees which represent more complex actions, and these actions can be composed into higher level behaviours
 - Task nodes and action sub-trees can be reused, and different sub-trees can be developed independently of each other

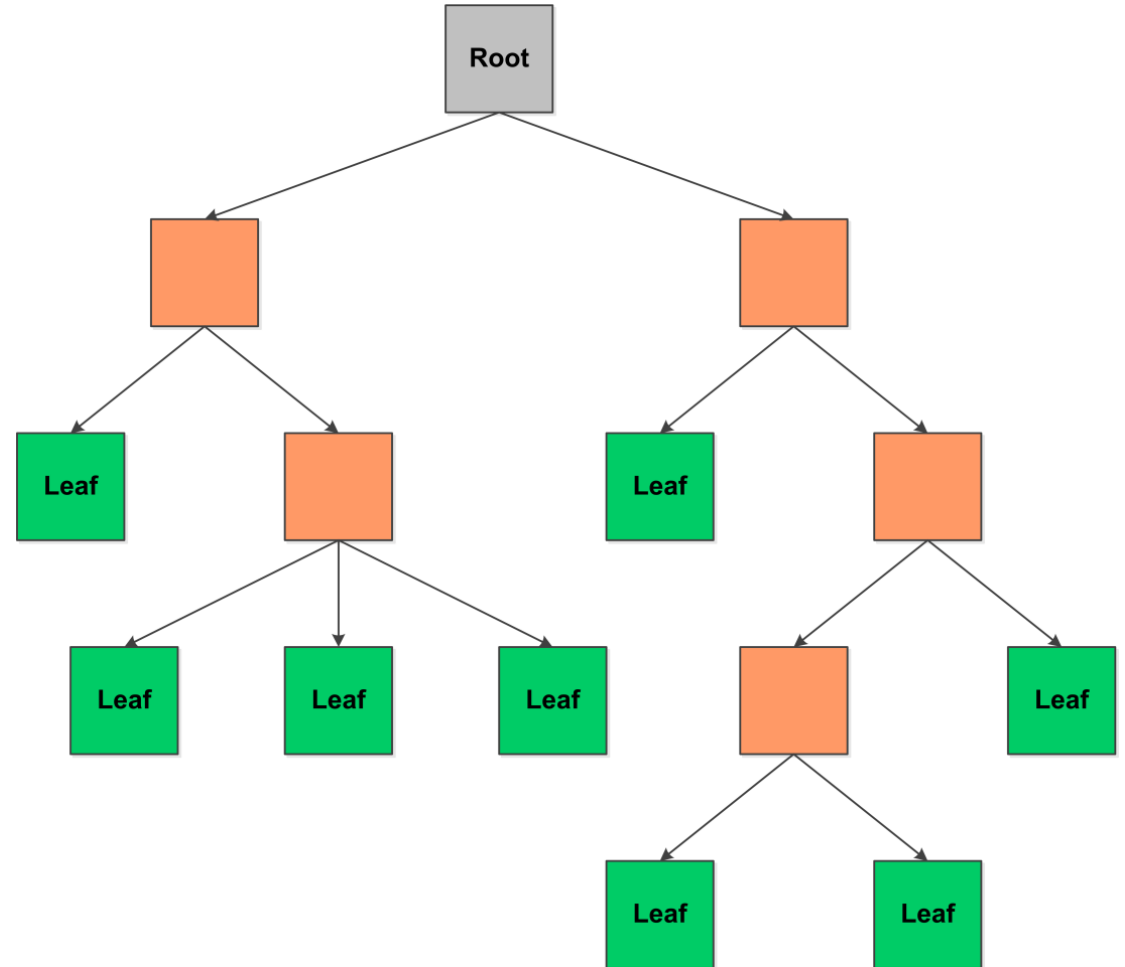
Structure

BTs are graphically represented as *directed rooted trees*:

- Composed of *nodes* and *directed edges*

A BT represents all the possible courses of action an agent can take:

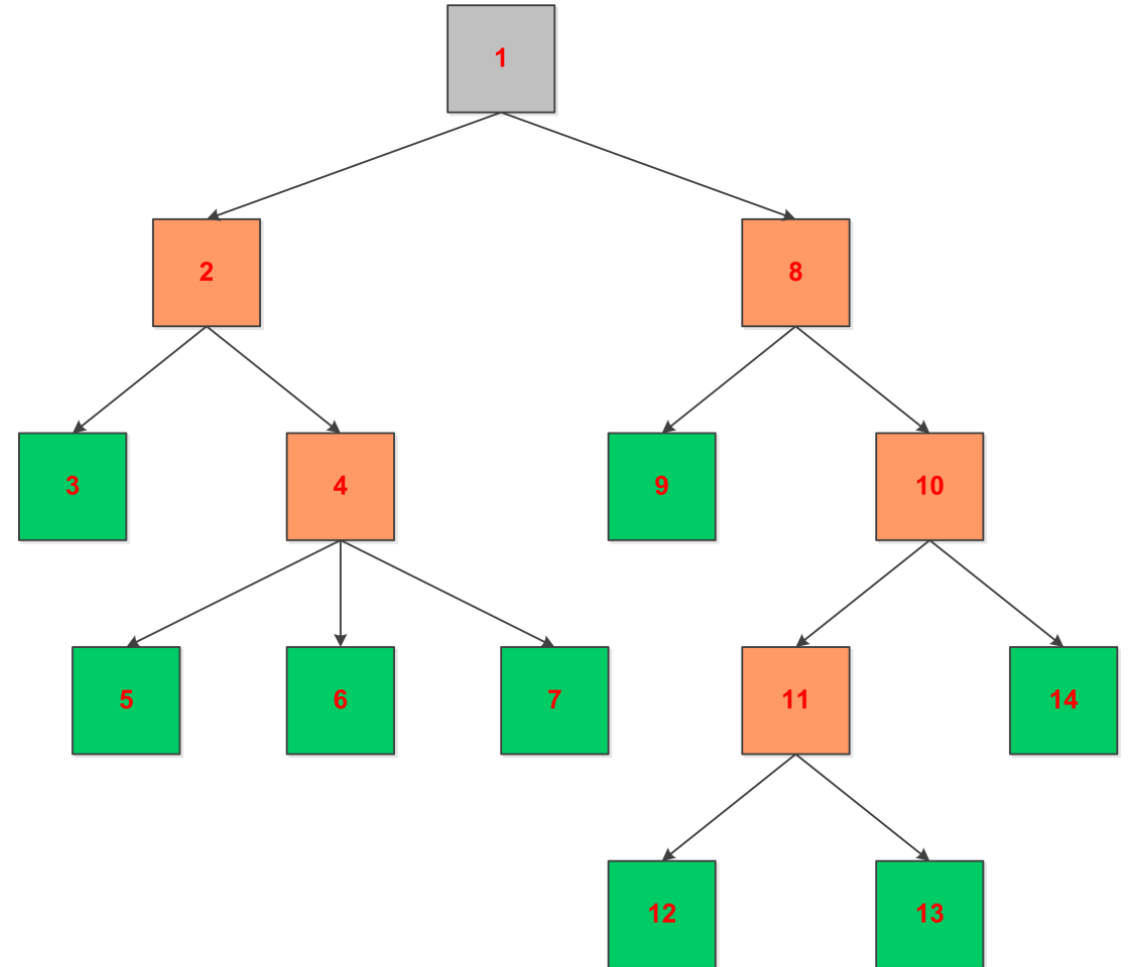
- A path from the root to one of the leaf nodes typically represents one possible course of action



Traversal

BTs are traversed in a *depth-first* manner (from left to right):

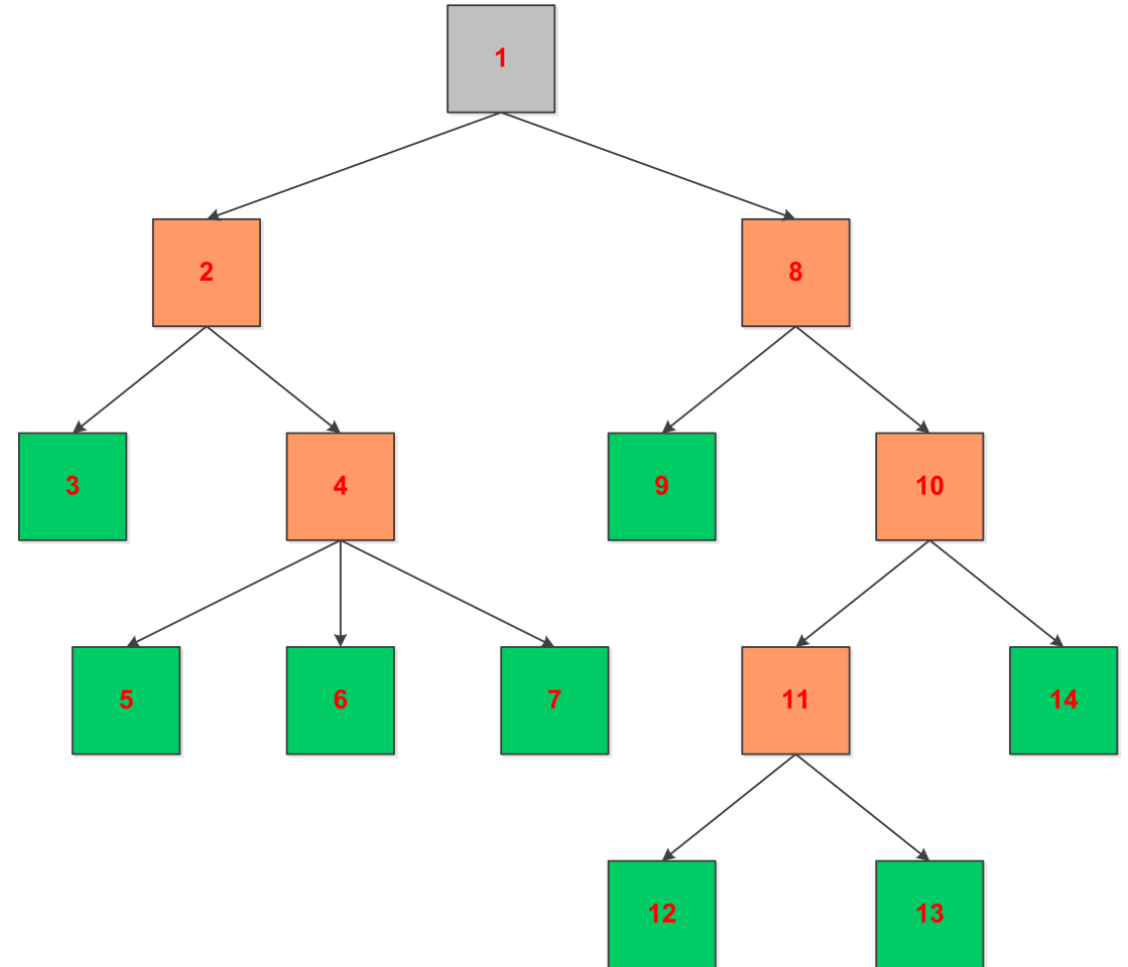
- An AI engine will usually traverse the BT from the root for each *simulation step* or *tick*, executing each node down the tree



Traversal

During the traversal 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





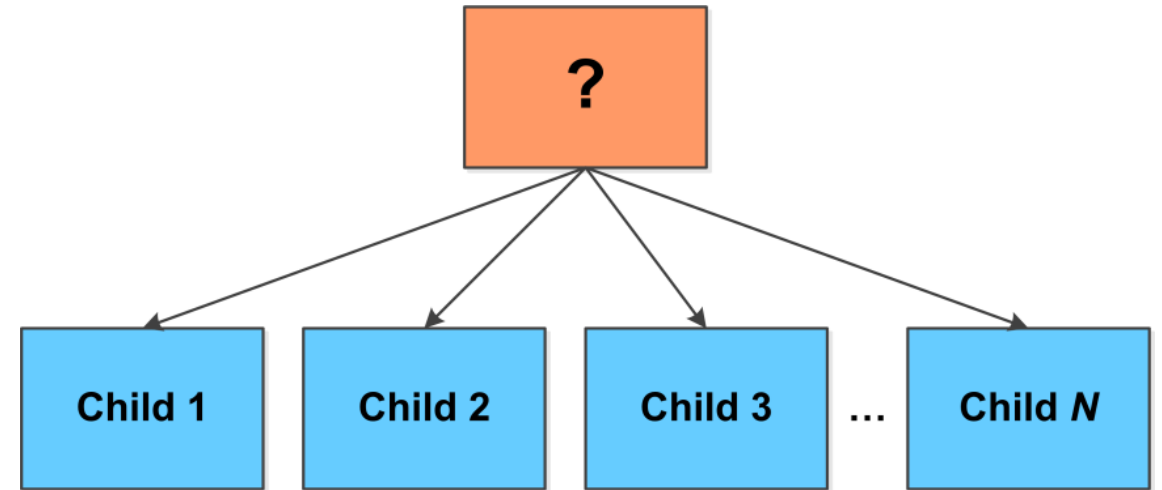
Categories of Nodes

- ***Control flow nodes*** (or composite nodes):
 - Interior nodes
 - Types: *selector nodes*, *sequence nodes*, and *parallel nodes*
- ***Task nodes*** (or execution nodes):
 - Leaf nodes
 - Types: *condition nodes* and *action nodes*
- ***Decorator nodes***:
 - Have only one child and modify the behaviour of the child in some way

Selector Nodes

Executes each of its children from left to right, and returns:

- **Success:** As soon as one of the children returns success
- **Failure:** If all the children return failure
- **Running:** If the child that is currently being executed returns running at the end of the simulation step

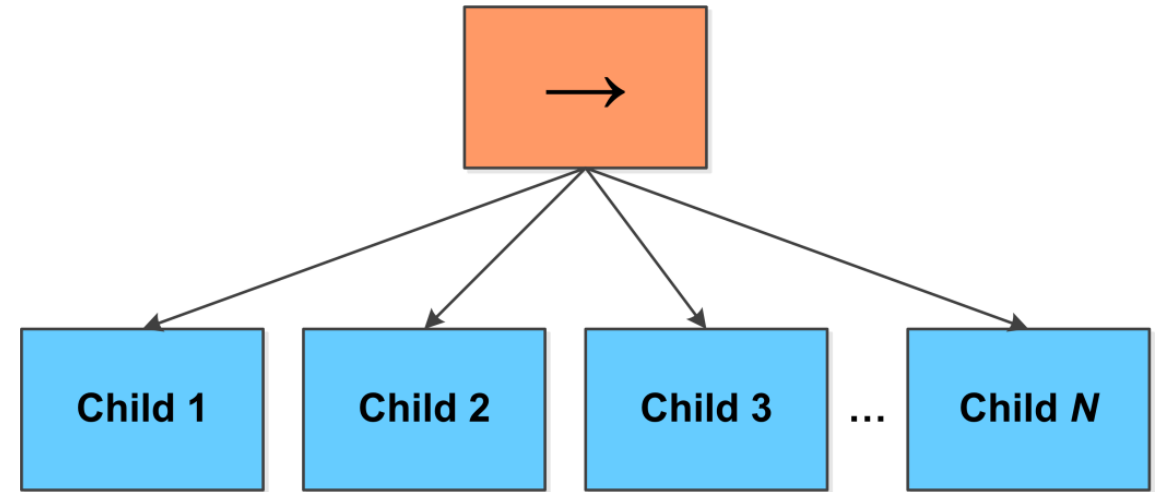


Selector nodes are typically used when a set of actions represent alternative ways of reaching a goal

Sequence Nodes

Executes each of its children from left to right, and returns:

- **Failure:** As soon as one of the children returns failure
- **Success:** If all the children return success
- **Running:** If the child that is currently being executed returns running at the end of the simulation step



Sequence nodes are typically used when a set of actions needs to be carried out in a particular order



Condition Nodes

Checks if a given condition within the simulated environment (or the real world) is fulfilled, and returns:

- **Success**: If the condition is fulfilled
- **Failure**: Otherwise
- **Running**: Never





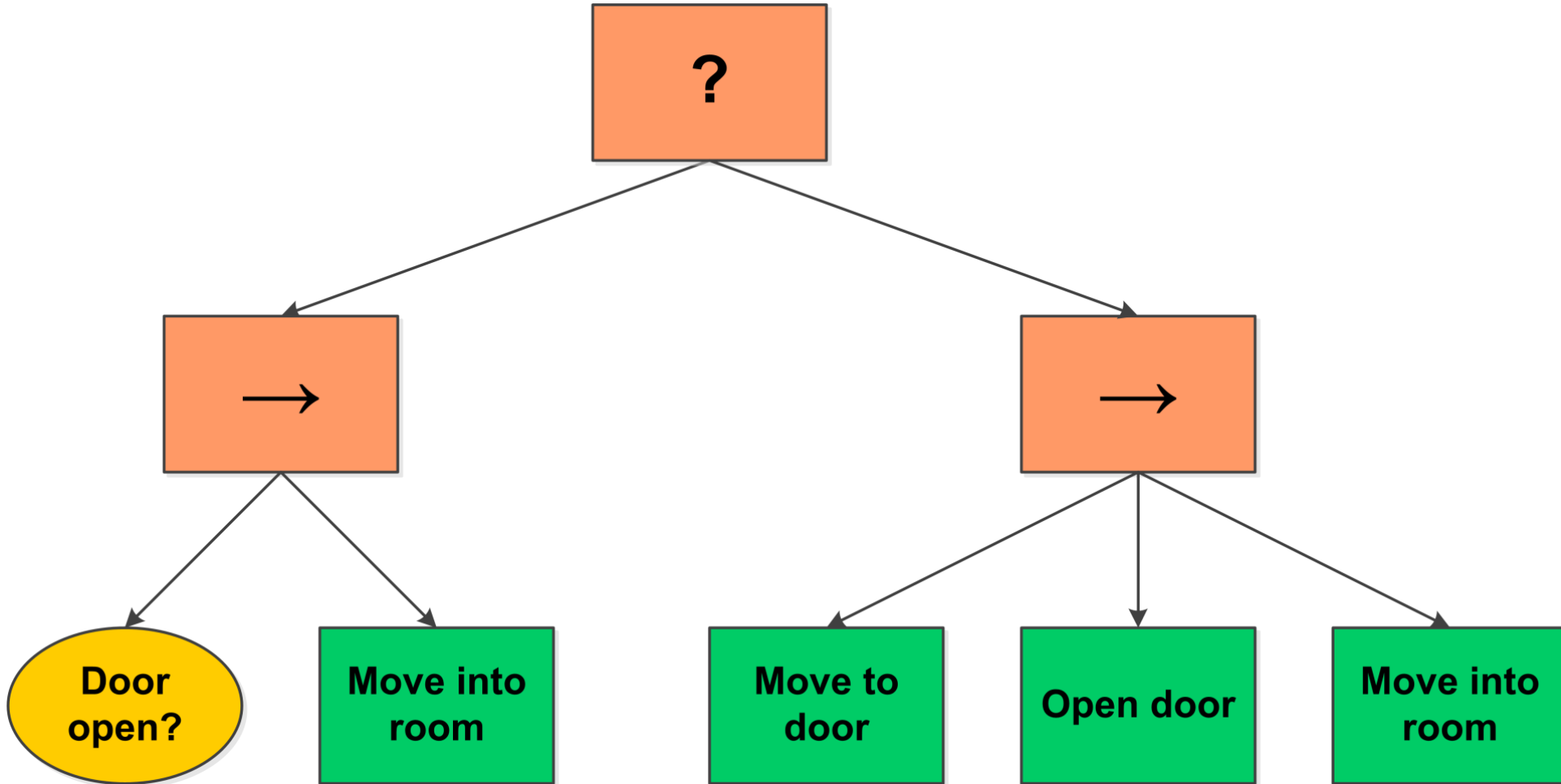
Action Nodes

Performs an action, which alters the state of the simulated environment (or the real world), and returns:

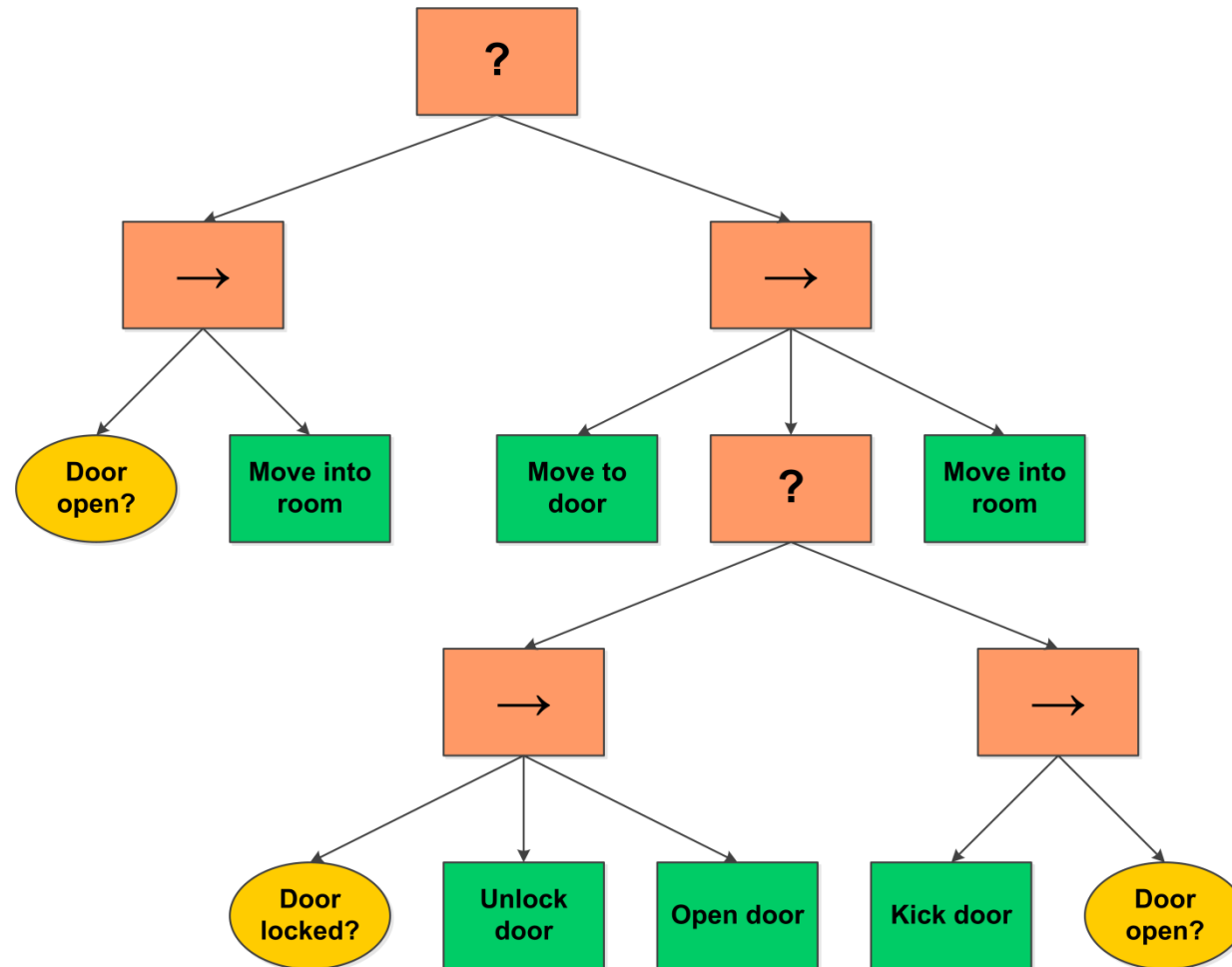
- **Success**: If the action was completed
- **Failure**: If the action could not be completed
- **Running**: If the action was not finished within the current simulation step



Example 1: Move into Room



Example 2: Move into Room (More Advanced)





Developing BTs

- 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
- In a BT the left branch of the tree will contain the high-priority behaviours, while the right branch will contain the low-priority behaviours
 - The default or unconditional behaviour will therefore be found at the far right side of a BT
- Most modularity is achieved if each task can be broken into the smallest parts that can usefully be composed
 - **Rule of thumb:** A BT should be decomposed into the smallest action nodes which do not have sub-parts that are likely to be used as stand-alone actions in other parts of the BT



Advantages

The most important advantages with BTs:

- Highly *composable* (ability to combine components into various combinations)
- Highly *modular* (can be subdivided into modules, and any module can be replaced by any other module)
- *Reactive* (react quickly to changes)
- *Human readable* and can be created by *visual editors*
- Suitable for *automatic generation* (for example by using machine learning techniques)



Limitations

The most important limitations of BTs:

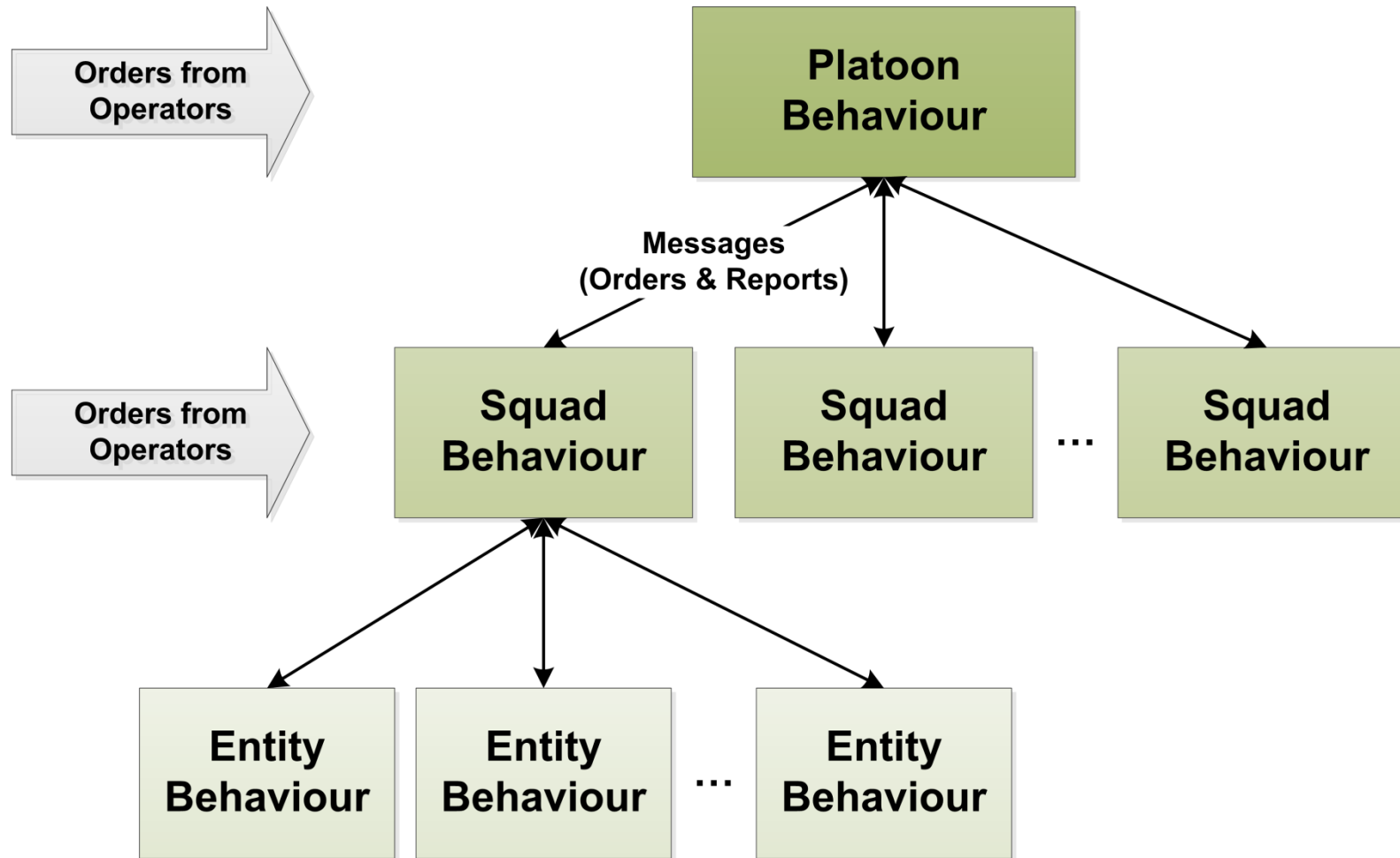
- Poor at modelling the uncertainty in situations where there are multiple valid options to choose from
- It is somewhat cumbersome to represent typical state-based behaviour using BTs
- For very large BTs the cost of having to execute the whole tree from the beginning for each simulation step will eventually cause performance issues, especially in simulations with a high number of constructive entities



Modelling Battle Drills

- We are currently building a BT-based library of behaviour models of the most important battle drills for mechanized infantry (dismounted soldiers and combat vehicles)
- The behaviour model library has a *hierarchical structure*:
 - Models of battle drills for *entities, squads* (only for dismounted soldiers) and *platoons*
- Human operators will give orders to the semi-automated forces (SAF) at the squad or platoon level
 - The entities will be completely autonomous within a squad (for dismounted soldiers) or platoon (for vehicles)
- Future work: build 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

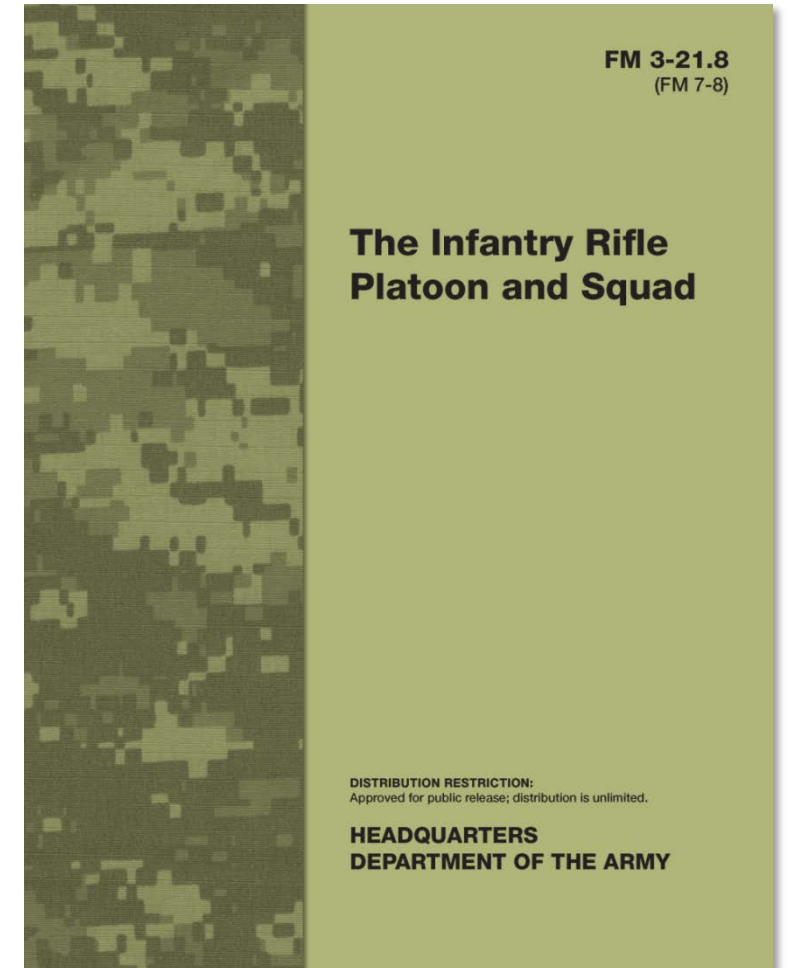
Modelling Battle Drills



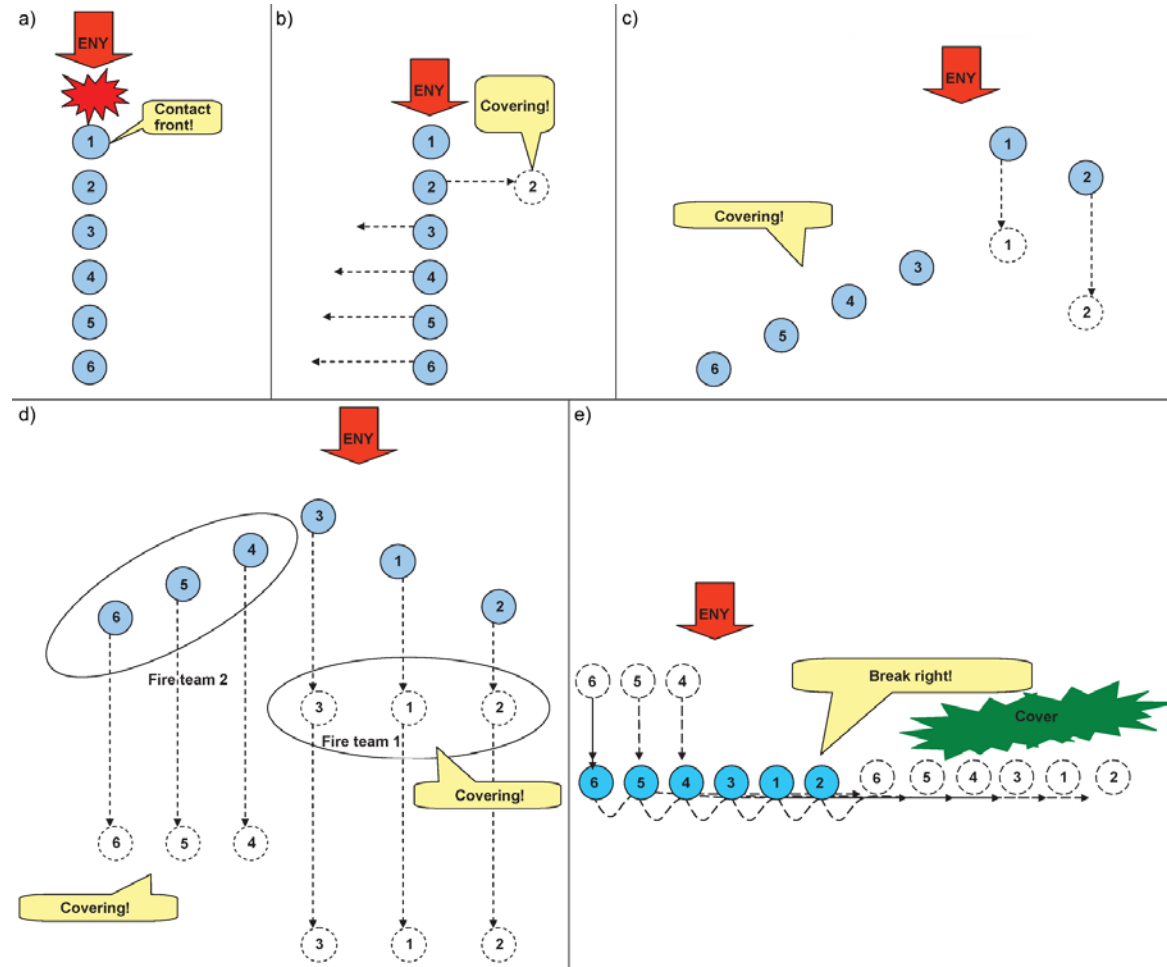
Modelling Battle Drills

Sources of information:

- Universal Task Lists (UTLs)
- Field manuals
- Subject matter experts (SMEs) and officers



Battle Drill for Enemy Contact for a Dismounted Infantry Squad







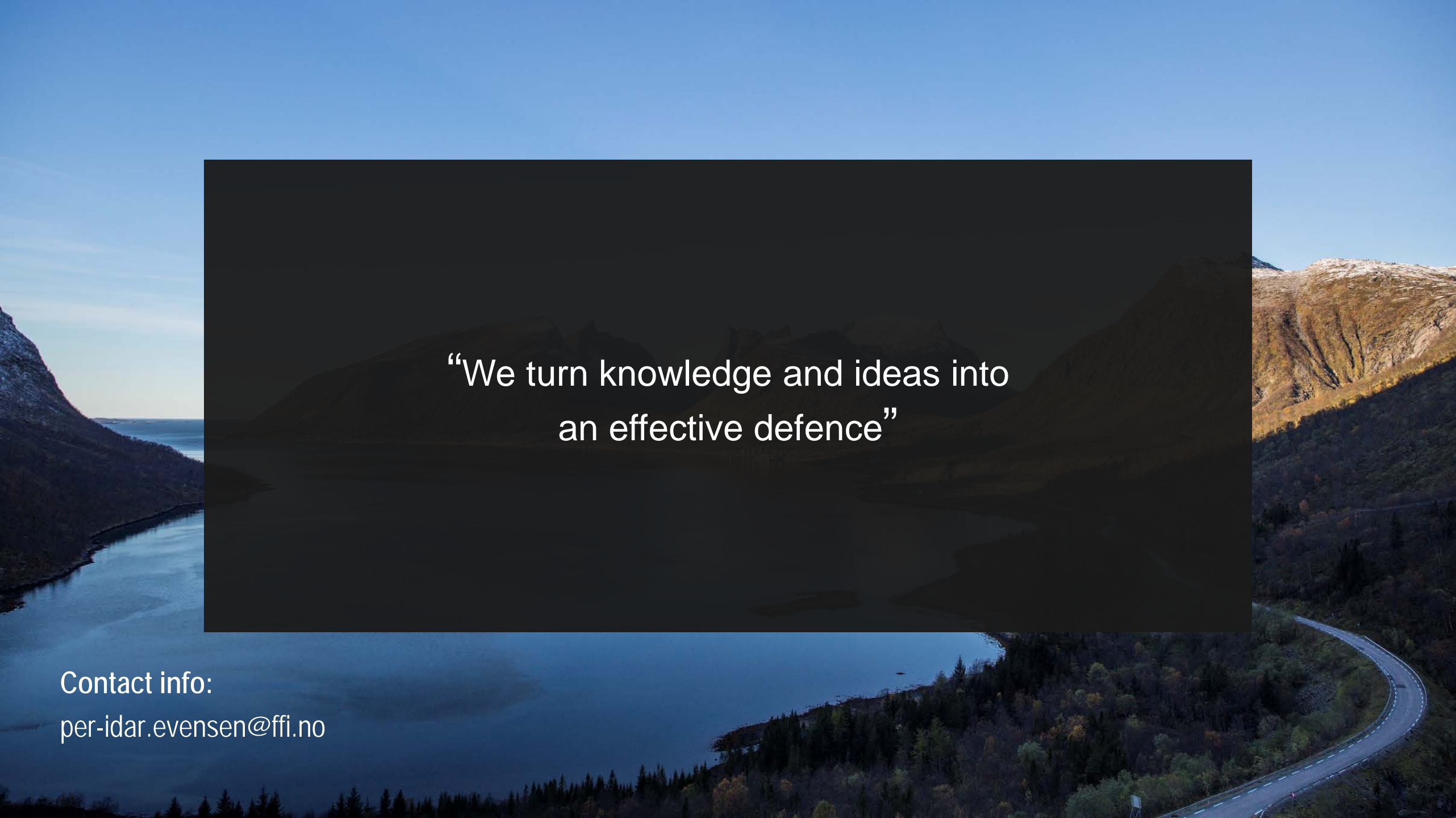
Experiences with BTs

- BTs have a somewhat steeper learning curve than for example Finite State Machines (FSMs)
- The composability and modularity that enables sub-trees to be reused is very useful and simplifies the development process
- A good visual editor with run time debugging functionality is very helpful for creating BTs



Summary and Conclusion

- BTs have become very popular, mainly because they are *composable*, *modular*, and *reactive*
- BTs are well suited for developing human behaviour models of moderate complexity for semi-automated forces (SAF) in constructive simulations
- The composability and modularity of BT-based behaviour models open up opportunities for collaboration on development and sharing of behaviour models of battle drills (e.g., between NATO and partner nations that mostly have similar doctrines)



“We turn knowledge and ideas into
an effective defence”

Contact info:

per-idar.evensen@ffi.no