# A survey of Behavior Trees in robotics and AI

Matteo Iovino *, Edvards Scukins, Jonathan Styrud, Petter Ögren, Christian Smith

*KTH Royal Institute of Technology, Stockholm, Sweden*

**ABSTRACT**

Behavior Trees (BTs) were invented as a tool to enable modular AI in computer games, but have received an increasing amount of attention in the robotics community in the last decade. With rising demands on agent AI complexity, game programmers found that the Finite State Machines (FSM) that they used scaled poorly and were difficult to extend, adapt and reuse.

In BTs, the state transition logic is not dispersed across the individual states, but organized in a hierarchical tree structure, with the states as leaves. This has a significant effect on *modularity*, which in turn simplifies both synthesis and analysis by humans and algorithms alike. These advantages are needed not only in game AI design, but also in robotics, as is evident from the research being done.

In this paper we present a comprehensive survey of the topic of BTs in Artificial Intelligence and Robotic applications. The existing literature is described and categorized based on methods, application areas and contributions, and the paper is concluded with a list of open research challenges.

## 1. Introduction

In this paper we survey the area of Behavior Trees (BTs) in AI and robotics applications (see an overview of the covered topics in Fig. 1).[1]

A BT describes a policy, or controller, of an agent such as a robot or a virtual game character. Formally, the policy or controller maps a state to an action. We illustrate the idea with a simple example of a BT, see Fig. 2. The actions are illustrated by gray boxes ("Evade", "Find aid", "Fire arrow at player", "Swing sword at player", "Taunt player", and "Wander"), while the state of the world is analyzed in the conditions, illustrated by white ovals ("Player is Attacking?", "Has low health?", and "Player is visible?").

We begin by listing a number of *key ideas* behind the design of BTs, while a detailed description of the execution is presented in Section 1.2.

- There is explicit support for the idea of *task hierarchies*, where one task is made up of a number of subtasks, which in turn have subtasks. In Fig. 2, *Fighting* is done either with a bow or a sword, and fighting with a bow in turn could include grasping an arrow, placing it on the string, pulling the string, and then releasing it.
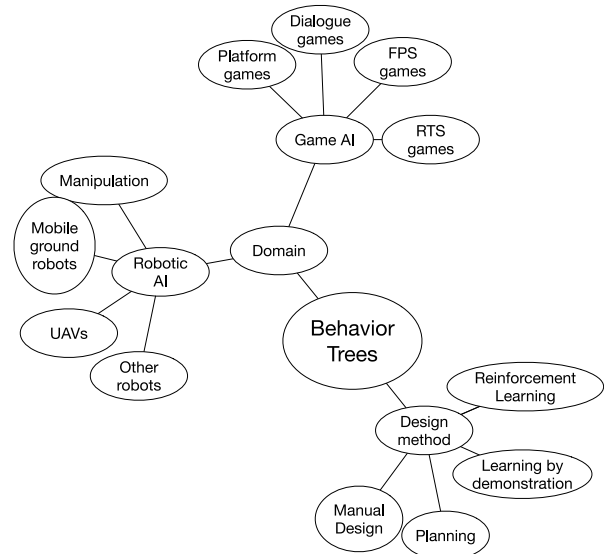


**Fig. 1.** Overview of the topics covered in this survey.

- There is explicit support for *Sequences*, sets of tasks where each task is dependent on the successful completion of the previous, such as first grasping an arrow and then moving it. Thus, the next item in a *Sequence* is only started upon success of the previous one.
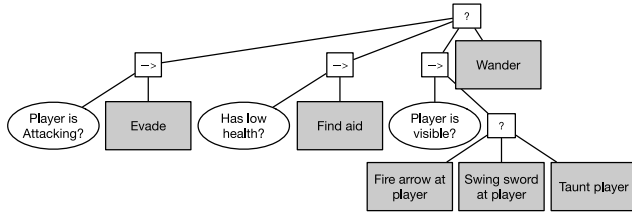
---

**Table 1**
The five node types of a BT.

| Node type | Symbol | Succeeds | Fails | Running |
|---|---|---|---|---|
| Sequence | → | If all children succeed | If one child fails | If one child returns running |
| Fallback | ? | If one child succeeds | If all children fail | If one child returns running |
| Parallel | ⇉ | If $\geq M$ children succeed | If $> N - M$ children fail | else |
| Action | shaded box | Upon completion | When impossible to complete | During completion |
| Condition | white oval | If true | If false | Never |



**Fig. 2.** An example of a BT for a *First Person Shooter* (FPS) game.
*Source:* Adapted from [3].

- There is explicit support for *Fallbacks*, sets of tasks where each task represents a different way of achieving the same goal, such as fighting with either a bow or a sword. Thus, the next item in a *Fallback* is only started upon *Failure* of the previous one.
- There is support for *reactivity*, where *more important tasks interrupt less important ones*, such as disengaging from a fight and finding first aid when health is low.
- Finally, *modularity* is improved by the fact that each task, on each level in the hierarchy, has the same interface. It returns either *success*, *failure*, or *running*, which hides a lot of implementation details, while often being enough to decide what to do next.

All items above are different aspects of the rules for switching from one task to another. By encoding this task switching logic in the hierarchical structure of the non-leaves of a tree, having basic actions and conditions at the leaves, a surprisingly modular and flexible structure is created.

Below we first give a brief description of the history of BTs, and then a more detailed description of their core workings.

### 1.1. A brief history of BTs

BTs were first conceived by programmers of computer games. Since important ideas were shared on only partially documented blog posts and conference presentations, it is somewhat unclear who first proposed the key ideas, but important milestones were definitely passed through the work of Michael Mateas and Andrew Stern [4], and Damian Isla [5]. The ideas were then spread and refined in the community over a number of years, with the first journal paper on BTs appearing in [6]. The transition from Game AI into robotics was even later, and independently described in [7] and [8].

### 1.2. Formulation of the core parts of a BT

A BT is a directed tree where we apply the standard meanings of *root*, *child*, *parent*, and *leaf* nodes. The leaf nodes are called *execution nodes* and the non-leaf nodes are called *control flow nodes*. Graphically, the BT is either drawn with the root to the far left and children to the right, or with the root on top and children below. We will use the latter convention, see the example in Fig. 2.

The execution of a BT starts from the root node, that generates signals called *Ticks* with a given frequency. These signals enable the execution of a node and are then propagated to one or several of the children of the ticked node. A node is executed if, and only if, it receives *Ticks*. The child immediately returns *Running* to the parent, if its execution is under way, *Success* if it has achieved its goal, or *Failure* otherwise.

In the classical formulation, there exist three main categories of control flow nodes (*Sequence*, *Fallback* and *Parallel*) and two categories of execution nodes (*Action* and *Condition*), see Table 1.

**Sequences** are used when some actions, or condition checks, are meant to be carried out in sequence, and when the *Success* of one action is needed for the execution of the next. The Sequence node routes the *Ticks* to its children from the left until it finds a child that returns either *Failure* or *Running*, then it returns *Failure* or *Running* accordingly to its own parent, see Algorithm 1. It returns *Success* if and only if all its children return *Success*. Note that when a child returns *Running* or *Failure*, the Sequence node does not route the *Ticks* to the next child (if any). In the case of *Running*, the child is allowed to control the robot, whereas in the case of *Failure*, a completely different action might be executed, or no action at all in the case where the entire BT returns *Failure*. The symbol of the Sequence node is a box containing the label "→".

---

**Algorithm 1:** Pseudocode of a Sequence node with $N$ children

1 **for** $i \leftarrow 1$ **to** $N$ **do**
2 　　$childStatus(i) \leftarrow \texttt{Tick}(child(i))$
3 　　**if** $childStatus(i) = running$ **then**
4 　　　　**return** *running*
5 　　**else if** $childStatus(i) = failure$ **then**
6 　　　　**return** *failure*
7 **return** *success*

---

**Fallbacks**[2] are used when a set of actions represent alternative ways of achieving a similar goal. Thus, as shown in Algorithm 2, the Fallback node routes the *Ticks* to its children from the left until it finds a child that returns either *Success* or *Running*, then it returns *Success* or *Running* accordingly to its own parent. It returns *Failure* if and only if all its children return *Failure*. Note that when a child returns *Running* or *Success*, the Fallback node does not route the *Ticks* to the next child (if any). The symbol of the Fallback node is a box containing the label "?".

**Parallel** nodes tick all the children simultaneously. Then, as shown in Algorithm 3, if $M$ out of the $N$ children return *Success*, then so does the parallel node. If more than $N - M$ return *Failure*, thus rendering success impossible, it returns *Failure*. If none of the conditions above are met, it returns running. The symbol of the Parallel node is a box containing the label "⇉".

**Action** nodes typically execute a command when receiving *Ticks*, such as e.g. moving the agent. If the action is successfully completed, it returns *Success*, and if the action has failed, it returns *Failure*. While the action is ongoing it returns *Running*. Actions are represented by shaded boxes.

---

2 Fallback nodes are sometimes also referred to as *selector* or *priority selector* nodes.

**Algorithm 2:** Pseudocode of a Fallback node with $N$ children

```
1  for i ← 1 to N do
2  │  childStatus(i) ← Tick(child(i))
3  │  if childStatus(i) = running then
4  │  │  return running
5  │  else if childStatus(i) = success then
6  │  │  return success
7  return failure
```

**Algorithm 3:** Pseudocode of a Parallel node with $N$ children and success threshold $M$

```
1  for i ← 1 to N do
2  │  childStatus(i) ← Tick(child(i))
3  │  if Σ_{i:childStatus(i)=success} 1 ≥ M then
4  │  │  return Success
5  │  else if Σ_{i:childStatus(i)=failure} 1 > N − M then
6  │  │  return failure
7  return running
```

**Condition** nodes check a proposition upon receiving Ticks. It returns *Success* or *Failure* depending on if the proposition holds or not. Note that a Condition node never returns a status of *Running*. Conditions are thus technically a subset of the Actions, but are given a separate category and graphical symbol to improve readability of the BT and emphasize the fact that they never return running and do not change the world or any internal states/variables of the BT. Conditions are represented by white ovals.

### 1.3. Overview

The purpose of this paper is to provide an overview of the current body of literature covering BTs in robotics. To do this, the Google Scholar,[3] Scopus,[4] and Clarivate Web of Science[5] databases were searched with the keyword "Behavior Tree" and alternative spelling "Behaviour Tree". Only papers written in English have been taken into account. This resulted in 297 papers. Removing the papers referring to Behavior Trees as functional requirements (as in e.g. [1]) and cleaning the list from dead links and duplicates, the final number as of April 24, 2020 was 166 papers. All papers are classified according to topic matter and application areas in Table 2, and this structure is also used throughout the paper.

The remainder of the paper is organized as follows. In Section 2 we review the papers that analyze the core theory of BTs. Then, in Section 3, we review the work done in the different areas of application. The different methods being used are then analyzed in Section 4. Finally, some common libraries for implementing BTs are discussed in Section 5, some open challenges are listed in Section 6, and the conclusions are drawn in Section 7.

## 2. Fundamental theory

In this section we describe the theoretical evolution of the BT core concept, including some extensions and tools for theoretical analysis. We will roughly follow the structure provided by Table 3.

Sequences were introduced in [4], as composed of behaviors that can either *succeed* or *fail*. There was no notion of returning *running* in this early version, an important concept that is needed to provide *reactivity*, as will be described below.

Fallbacks were also present in an early form in [4], where it is described how actions to achieve a given result are collected, and their corresponding preconditions are checked. If more than one are satisfied, the one with the highest specificity is chosen, see Fig. 3. Fallbacks were subsequently given a more refined form in terms of the prioritized lists in [5] and [6].

*Reactivity*, the ability to stop an action before it completes (either *succeeds* or *fails*) and switch to a more urgent action, is a very important quality of an agent. In modern BTs this is achieved using the *running* return status, in combination with recurrent ticks from the root, prompting a reevaluation of conditions and actions, see Algorithms 1–2. It was suggested in [6] that "conditions are reevaluated after a given number of game ticks (frames), when certain game events occur or when the active behavior terminates". In the formulation described in Section 1.2, reevaluation takes place when the tick is sent from the root with a given frequency, to provide *reactivity* to external events. If external changes can somehow be tracked centrally, the same *reactivity* would be achieved by only ticking "when certain game events occur". However, only ticking "when the active behavior terminates" is vastly different, as it would imply that the agent always finishes its current task before starting a new one, effectively preventing any reactions to unexpected high priority events such as fire alarms, safety margin infractions, or the appearance of other threats or opportunities. In fact, the absence of a reactive capability was pointed out as the main drawback of BTs in [54], where a BT formulation without *running* was used. This is somewhat ironic since today *reactivity*, together with *modularity*, is often listed as some of the main advantages of using BTs. This use of *Ticks*, and the *running* return status as described in Algorithms 1–2, including the Parallel node of Algorithm 3, was described in [157].

Prior to BTs, the Finite State Machine (FSM) has long been used as a way of organizing task switching, and a BT can be seen as a FSM with special structure, as noted in [5]. The relationship between FSMs and BTs was further explored in [7,95,102,159]. In these papers, it was shown how to create a BT that works like an FSM, by keeping track of the current state as an external variable on a *blackboard*, and an FSM that works like a BT by letting all ticks and return statuses correspond to state transitions. Thus, the two structures are equivalent in terms of what overall behavior can be created, much in the same way as an algorithm can be implemented in any general programming language. However, from a practical standpoint, the differences are often significant.

FSMs use state transitions, that transfer the control of the agent in a way that is very similar to the GOTO statement, that is now abolished in high level programming languages, [161]. In contrast, the control transfer in a BT resembles function calls, with subtrees being ticked, and returning *success*, *failure* or *running*. Furthermore, assuming $n$ possible actions (represented as states), the $n^2$ possible transitions of an FSM rapidly turns to a single large monolithic structure that is very complex to debug, update and extend [56]. For a BT on the other hand, each subtree is a natural level of abstraction, with all subtrees sharing the same interface of incoming ticks and outgoing return statuses.

Beyond the FSMs, the relationship between other switching architectures and BTs have been explored. It was shown in [102] how Decision Trees and the Subsumption Architecture are generalized by BTs and in [158] how the Teleo-Reactive approach [162] and AndOrTrees are generalized.

The standard Fallback node in Algorithm 2 has a static priority order of all its children, and it has been noted by several authors

**Table 2**

Taxonomy of Behavior Trees papers.

| Methodology | Applications | | | | | |
|---|---|---|---|---|---|---|
| | Game AI | Combat training simulation | Robotics | | | Others |
| | | | Manipulation | Mobile ground robots | Aerial and underwater vehicles | |
| Learning by RL/GP | [9–27] | [28–35] | | [36–40] | [41] | [42,43] |
| Learning by Demo | [44–51] | | [52] | [52] | | |
| Hand coded | [3,5,6,44,53–89] | [90–94] | [8,62,95–104] | [63,95,103,105–118] | [7,63,106,119–128] | [42,129–131] |
| Planned analytically | [132,133] | | [96,134–137] | [134–136,138–140] | [141] | [142] |
| Others | [143–150] [4,151–153] | [154] | [155] | [155] | | [156] |

**Table 3**

Areas of fundamental theory for BTs.

| Fundamental theory area | Papers |
|---|---|
| Establishing the BT core | [4–6,157] |
| Architecture comparisons | [5,7,95,102,158,159] |
| Variations on Fallback node | [60,110] |
| Convergence analysis | [102,121,136,137,158] |
| Stochastic analysis | [62,110,160] |
| Analysis of Parallel node | [97,103,104,107] |
| Parameters and data passing | [55] |
| Multi agent BTs | [106] |
| Learning BTs | See Section 4.1 |
| Planning BTs | See Section 4.2 |

that there are many cases where it makes sense to update this order based on the state of the world. One such approach, Utility BTs, was suggested in [60]. Here each subtree of the BT is able to return a utility value, and there are special *Utility Fallbacks*, that sort their children based on this utility score. Leaf nodes then have to implement such a utility estimate, whereas other interior nodes have to be able to aggregate utility based on their children. This can be done in many ways, and the one suggested in [60] is to report the highest child utility as the utility of both Fallbacks and Sequences.

Another way of addressing static Fallbacks, based on estimated success probabilities, was suggested in [110]. Here the system gathers data on the *success/failure* outcomes of all leaves. Thereby it is able to estimate the success probability of each leaf, and reorder the children of a Fallback accordingly. This result is data driven and may therefore also be considered to be a learning approach. Such approaches are described in detail in Section 4.1.

Given such statistics on *success* and *failure* probabilities of leaves, it is natural to expand to *success* and *failure* probabilities of entire subtrees, as in [62]. In addition, if given estimates of execution time of the leaf nodes, in terms of probability density functions over time for success and failure, these estimates can be aggregated upwards and render execution time estimates for all subtrees. Finally, the stochastic analysis of BTs are taken one step further in [160], where Hidden Markov Models (HMM) are connected to BTs where only noisy observations are available from the execution. Given observation data, it is shown how HMM tools can be applied to estimate state transition probabilities of the BT, to estimate what transitions are the most likely given some data, and how likely the output of some data is, given a set of parameters.

The Sequence node in Algorithm 1 is reactive in the sense that it constantly reevaluates its children. This can sometimes cause problems when executing actions that do not leave a trace of their successful completion. An example of this was described in [120] with an agent following a waypoint path. After passing a waypoint the agent switches to the next. However, if the success condition is the distance to the waypoint, the first action will not return *Success* when leaving the first waypoint on its way to the second. One common solution to this problem is to create a new node type, which is a Sequence node with a memory[6] of what child is active, and skip over a child that has already returned *Success* at some earlier point in time. However, as noted in [120] the same result can be achieved by adding a *decorator* node above each waypoint action that keeps returning *success* without ticking its child, after the initial *success* has been returned.

A reasonable question to ask is in what parts of the state space a BT will return *success* or *failure*. This problem was addressed for Teleo-reactive designs in [162] and carried over to BTs in [102] and [97]. In order to do a formal statespace analysis, a functional model was proposed in [63,102]. Using tools from control theory, such as *region of attraction*, and *exponential stability*, properties such as robustness (regions of attraction), safety (avoidance of some regions) and efficiency (convergence within upper time bounds) were addressed. By showing how the properties carry over across sequence and fallback compositions, the analysis can be done for larger BTs.

The analysis of robustness, safety and efficiency was continued in [42], where it was shown how a learning subtree, with possibly unreliable performance, could be added while some guarantees of robustness, safety and efficiency could still be given for the overall BT.

In [137], a concept that is very similar to BTs was introduced in the form of Robust Logical Dynamical Systems. Convergence of Sequences of such components was proven, together with a stochastic analysis of convergence times when faced with a bounded number of external state transitions.

[136] extend the core notation of BTs by adding pre- and post-conditions to the Action nodes and thus removing Condition nodes. This modified model is called *extended Behavior Tree* (eBT) and is meant to be interfaced with Hierarchical Task Network (HTN) planning. This planning algorithm is also responsible for rearranging the tree, optimizing the execution time and the resources used, but without compromising the correctness of the tree. The authors also state the differences between their model

---

6 sometime called Sequence*

**Table 4**
BTs for Game AI.

| Type of game | Papers |
| --- | --- |
| Real time strategy games (RTS) | [12,14,18,24,45,46,49,82,151,152] |
| First Person Shooters (FPS) | [5,6,47,50,51,58,75] |
| Platform games | [10,11,13,16,17,57,70], |
| Dialog games | [4,53,65,69,72,77,163] |

and the standard one. This framework is also used in [97], where it is combined with a *Motion Generator* (MG), responsible for superposing individual and independent motion primitives to generate a hybrid motion. This setup allows the activation of different primitives concurrently, through the Parallel operator, hence crossing the limits stated also in [107].

Most actions of a BT require data on the world, such as positions of objects and other agents, or the amount of charge left in a battery. [55] introduce parametrization in Behavior Trees. Instead of using a *blackboard*, i.e., a repository of data available to all the nodes of the tree as in [97], they propose to add parameters to a node such that the node stores all the parameters needed for its subtree execution. This is done in three different ways: it can be hard coded, taken from the characteristics of the world and the agents, or satisfied with *Parametrized Action Representation* (PAR) arguments, which allow for reuse and encapsulation of the sub-tree.
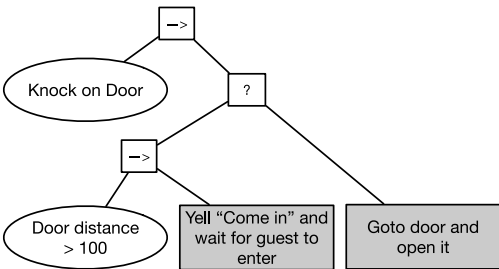
[107] introduce the notion of Concurrent Behavior Trees (CBT) as a way of improving the Parallel operator in standard BTs. Indeed, the Parallel operator is normally used when the sub-tree tasks are orthogonal, i.e., independent. However, this is not the case for most robotic applications, and the use of the Parallel operator can help mitigating the curse of dimensionality. The standard Parallel operator inherits the well known problems of parallel computing, namely *process synchronization* (when one or more processes are queued and the latter ones have to wait for the end of the execution of the former) and the *data synchronization* (when regulations is needed on data access). Therefore, the CBT is equipped with a *Progress Function*, which indicates the state of the BT execution, and a *Resource Function*, which lists a set of resources needed for the execution at each state. Finally, two new Parallel operators are defined. The *Synchronized Parallel operator*, stipulates that all the children share a minimum progress value that they have to reach in order to proceed with the execution, hence waiting for the slowest one. The *Mutually Exclusive Parallel operator* only executes a child if it uses resources which are not needed by other running children. These new operators were tested on a mobile robot. The parallel node was also discussed and expanded by [104] in combination with event driven nodes, in the context of an architecture for industrial robot control based on BTs.

## 3. Applications

In this section we describe different application areas, see Fig. 1, and give a short overview of how BTs have been used in the different areas (see Tables 4–6).

### 3.1. Game AI and chatbots

As described in Section 1.1, BTs were first created for a dialog game AI in [4], see Fig. 3, and have since become a standard tool for developing the AI of non-player characters in many different game genres, from strategy games to shooters. Below we discuss the use of BTs in these different game categories in more detail.



**Fig. 3.** A BT from a dialog game, describing how to react when someone knocks on the door.
*Source:* Adapted from [4].

**Table 5**
BTs for different kinds of robot systems.

| Robotic system | Papers |
| --- | --- |
| Manipulators | [8,62,96–102,104,134,137,155] |
| Mobile Manipulators | [52,103,106–109,116,134,136,139,140] |
| Wheeled Robots | [40,111–114,118] |
| Robot Swarms | [36–39,115,138] |
| Humanoid Robots | [95,102,110,117,135,155] |
| Autonomous vehicles | [63,105] |
| Aerial Robots | [41,119,120,122–125,128,164,165] |
| Underwater Robots | [121] |

**Table 6**
Other applications.

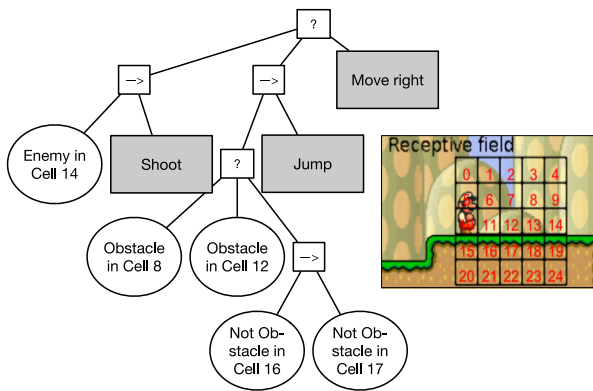| Area | Papers |
| --- | --- |
| Workflow and human activity modeling | [129–131,166–169] |
| Camera Control | [170,171] |
| Smart homes and Power Grids | [172–175] |
| Engineering Design Tasks | [176,177] |

### 3.1.1. Real-time strategy games (RTS)

Real-time strategy games require players to make tactical decisions, such as taking control of some given area, or aim an attack on some particular enemy facilities. In such games, there is a large number of different units, that can be assigned to specific tasks based on their different strengths and weaknesses [49]. These units can also be combined into squads, platoons and so on up to an entire army, and BTs have been used to make decisions on many such levels of abstraction [82]. An early paper on RTS AI is [12], where evolutionary methods were used to create BTs that could defeat a hand coded AI for the RTS game DEFCON. Other examples of evolutionary methods to create BTs include [18], [14], where an AI for turn based strategy games was developed.

In [45], the authors generated BTs based on recurring actions executed by human experts. The proposed approach did not need any action model or rewards, but required a significant number of training traces. An alternative approach using manually injected knowledge was proposed in [46]. One of the most well known, and complex, RTS games is StarCraft. In [151], StarCraft was used to demonstrate the ability of A Behavior Language (ABL) to do reactive planning, and a human-level agent was proposed in [152].

### 3.1.2. First person shooters (FPS)

One of the first papers on BTs describes how they are used to create the AI of the highly popular FPS game Halo 2 [5]. In FPS games, the player interacts with a large number of Non-Player

**Fig. 4.** An example of a BT for a platform game, that was created using an evolutionary learning approach. The cell numbers of the conditions can be found in the illustration to the right.
*Source:* Adapted from [17].

Characters in a highly dynamical combat environment, and an example of the decisions made and actions taken can be seen in Fig. 2.

The problem of reusing BTs in situations where the available actions of an agent increase over time was studied in [50]. An approach where actions are categorized is proposed, in which the leaves of the BT make high level queries to find the action suitable for the situation at hand. This work is then continued in [6,86], where a higher abstraction layer is added above the BT. Having a set of BTs for achieving different goals, case based reasoning is used to chose the appropriate BT to invoke.

Additional modifications have been applied to FPS BTs by including time, risk perception, and emotions [58], or by learning BTs from traces that were obtained during human game play (programming by demonstration) [47,51].

*3.1.3. Platform games*

Platform games are games where the player controls a character in a 2D environment and moves between a set of platforms [178]. The Mario AI framework emulates a popular platform game, and has become a widely used tool to test evolutionary approaches to learning a BT [10,11,13,17] (see Fig. 4). Pac-Man is not technically a platform game, but shares the 2D structure, and has been used to test Monte Carlo Tree Search using BTs [70], hybrid evolving BTs [16] and tools with focus on supporting manual BT design [57].

*3.1.4. Dialog games*

In an effort to make game environments more realistic, designers sometimes enable non-player characters to interact with the player not only through actions, but also through spoken dialog. This was the motivating application for the conception of BTs in [4], and has later been explored in [53,72,77,163]. In [53] situations such as a waiter making conversation while retrieving orders is considered, while [163] investigates a human–robot interaction scenario. Interactive narratives are explored in [77], where two virtual teddy bears ask the player to find them a ball to play with, while in [72], conversation types such as buyer and seller negotiations and simple asking–answering are investigated.

*3.2. Robotics*

The transition of BTs from Game AI into robotics was independently proposed and described in 2012, in [7,8]. The former applies BTs to object grasping and dexterous manipulation, and the latter proposes to use BTs for UAV control. In this section we divide the papers into three categories based on the type of robot that BTs are applied to: *manipulators*, *mobile ground robots*, and *aerial and underwater robots*. A division into even finer categories can be found in Table 5.

*3.2.1. Manipulation*

Since 2012 [8], BTs have been used in academia for control of robotic arm manipulation tasks, where they have proved to be useful, mainly due to the key characteristics of *transparency* and *modularity*. An example of a BT for controlling a mobile manipulator can be found in Fig. 5.
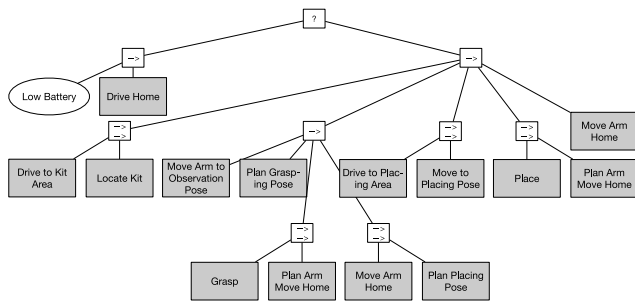
Within industrial robotics, BTs can be used to improve the *reactivity* of systems, which is especially important for collaborative robotics. One example of this is the commercial software platform Intera[7] of Rethink Robotics, now owned and developed by HAHN Robotics.

Other hardware setups with collaborative robots used in publications on BTs include the ABB YuMi [104,134], various versions of KUKA collaborative robots [97,99,134,136], the Universal Robots UR5 [98–101,136,155], the Softbank NAO [95,102,135, 155], and the Franka Emika Panda [137]. There are also some examples using other robots such as the Barrett WAM [8] and the IIT R1 robot [103].

The majority of the BT implementations used for robotic manipulation have been constructed manually but there are also examples using *Linear Temporal Logic* [135] and [96], *Backchaining* and STRIPS planners [134], *Planning Domain Definition Language (PDDL) planners* [136] and the *A\*-like planner* of [137]. Most of these papers use robotic manipulation as the main application, but the results are often applicable to robotics in general. Simulated systems are used in [62,103,134,136,137], and real ones in [8,95,97–102,104,135–137,155].

Looking at the timeline, the first paper using robotic arms [8] made a general proof of concept of task structuring with perception, planning and grasping for a Barrett WAM. [95] introduced a formal description of BTs and demonstrated grasping and transporting objects with NAO robots. [62] described how to compute performance measures for stochastic BTs for a robotic search and grasp task in simulation. [135] described a way to interface BTs with Linear Temporal Logic which was then demonstrated with NAO robots grasping and transporting objects. A series of papers, [98–101] used the software COSTAR, with a graphical interface for creating BTs. The effectiveness CoSTAR was then demonstrated on a number of systems. [98] used CoSTAR with a Universal UR5 performing a kitting task and a machine tending task, both with manually created BTs. [99] extended [98] with experiments on a KUKA LBR iiwa and UR5 robot performing tasks such as wire bending, polishing, and pick and place. [100] and [101] presented usability studies where users created BTs for UR5 robots in CoSTAR to perform pick and place tasks. They concluded that users "found BTs to be an effective way of specifying task plans". [136] presented extended BTs (eBT), which also include conditions to enable optimization which was shown to reduce execution time on a robotic kitting task. The demonstrations were performed on a KUKA LWR4+ on a mobile platform in simulation and on a real stationary Universal Robot UR5. [97] built on [136] and described using parallel superimposed motion primitives to generate complex behaviors and used Allen's interval algebra to define constraints in spatial relations demonstrated on a robotic assembly task with a KUKA iiwa 7 R800. [104] used a version of BTs with support for event handling to demonstrate real assembly of a planet gear carrier with the industrial ABB YuMi robot. [102] mainly showed theoretical results of how BTs modularize and generalize other architectures. These properties

---

7 https://www.rethinkrobotics.com/intera

**Fig. 5.** A BT for a mobile manipulator to pick and place objects in a kitting task. The BT was optimized for completion time by applying an algorithm that added Parallel nodes where possible.
*Source:* Adapted from [136].

were demonstrated on NAO robots grasping and moving balls. [134] described how to use a planning algorithm as a basis to iteratively and reactively generate BTs. The generated trees were demonstrated in two simulated scenarios. The first consisted of a KUKA Youbot doing pick and place tasks in the presence of obstacles and external agents. The second scenario was an ABB YuMi robot performing a simplistic version of cell phone assembly. [103] defined and analyzed two synchronization techniques to handle concurrency problems with BTs using parallel nodes. The techniques were demonstrated on simulated IIT R1 robots in four experiments with different navigation and object pushing and manipulation tasks. Finally, [137] drew inspiration from BTs and introduced a framework called Robust Logical Dynamical Systems (RLDS), shown to be equivalent to BTs in the sense that any BT can be descripted as an RLDS and vice versa. It was shown together with a simple planner to produce robust results for a Franka Emika Panda robot grasping objects and placing them in a kitchen drawer.

### 3.2.2. Mobile ground robots
Here we describe the application of BTs for mobile ground robots. The first work in this area was [63,95], and an example of a BT for mobile manipulation can be found in Fig. 5.

The *reactivity* provided by BTs is useful for providing ground robots with responses to detected faults or unknown obstacles [103,106,134,139], and the advantages are similar to the ones found in manipulator control, especially for mobile manipulators, as there is significant overlap between the topics.

In swarm robotics, a number of studies have been made on the combination of BTs and evolutionary algorithms [36–39,115,138, 179]. The tree structure of BTs gives good support for straightforward implementation of both mutations and crossover, while the *modularity* increases the chances of such variations resulting in functional designs.

BTs are used to perform search and rescue tasks and navigation in cluttered environments in [106,107]. Multi-robot systems are investigated in [106], where the *Parallel* node is used to improve the inherent fault tolerance of multi-robots applications. Mission performance is formally analyzed in terms of *minor* and *major* faults, where the robot is not able to perform some tasks or any tasks at all, respectively.

Fault detection is also a key characteristic of the framework proposed in [139] where the BT, used to control a simulated mobile manipulator, frequently checks the correspondence between the expected logic state of the world and what is sensed, allowing for replanning if discrepancies are found. Finally, in [103], a mobile manipulator (the R1 robot from IIT in Genova) is used as a proof of concept to illustrate how the execution of a parallel node can be synchronized or un-synchronized. The same robot

is used in [116] in an example of a task solved by the proposed Conditional Behavior Tree.

The *modularity* of BTs is crucial in applications like mobile manipulation, where the grasping capabilities of a manipulator are enhanced by a mobile platform. In particular, BTs of that kind are designed to reach a goal, subject to the condition of moving in a collision-free trajectory. This task is simulated with a KUKA Youbot in [134], as a practical proof of the scalability of automatically generating BTs for solving complex problems. A robot similar to the KUKA KMR iiwa is used in [136] to show that a BT, combined with an algorithm optimizing a skill sequence generated by a PDDL planner, can save 20% of the execution time, compared to a standard sequential execution. Similarly, in [140], a dual-arm mobile manipulator takes user's directions as audio input and uses automatic generated BTs to perform a picking task. In [52] a mobile manipulator performs a household cleaning task where navigation is required, using a BT learned from experience, as an alternative to the Decision Trees used in previous works. The paper uses BTs as a policy representation for the task because of its *modularity*, *transparency* and responsiveness. In [108,109] BTs are used to model a person following task, executed by a Toyota HSR (Human Support Robot). Here, the responsiveness of the BT is fundamental for coordinating sensor data with robot skills, to react to the changes of a dynamic environment.

BTs are combined with Semantic Trajectories (a trajectory enriched with context information) in [113] and [114]. This framework is applied to the navigation of a X80SV mobile robot performing a patrol task in simulation. BTs are used to implement the decision making algorithm, which uses a scenario defined by a set of context information provided by the semantic trajectories. In these papers, the authors show that the trajectory enriched framework improves the task execution of the autonomous agent. In [40], a simulated iRobot Create 2 performs a package delivery task. Here, Reinforcement Learning is used to learn a policy which is then automatically converted into a BT, which is stated to "offer an elegant solution […to …] the need for uniform action duration in both planning and plan recognition". Another example is portrayed in [118], where the authors present *Navigation2*, a ROS2 realization of the ROS Navigation Stack, which features configurable BTs to handle planning, control, and recovery tasks. The framework is tested in dynamic environments using TIAGo and RB-1 robots.

BTs have also been applied in decision making for soccer robots in [111,112], to control wheeled holonomic robots in the RoboCup competition. In particular, [111] combines the decision output of the BT (e.g. whether to pass, shoot or move) with a RRT path planner to achieve robot movement. In [112], the same authors combine BTs with Fuzzy logic and use Fuzzy membership functions as the leaves of the BT. A Fuzzy obstacle avoidance algorithm is then used for path planning. Experiments show that the Fuzzy logic obstacle avoidance algorithm has a better performance in terms of time and length of the computed path, when compared to other planners based on RRT or A*. They emphasize how the *modularity* of BTs "allows to easily extend decision making for complex tasks" and to simplify maintainability.

In Swarm Robotics a multi-robot system composed of a large number of homogeneous robots re-creates behaviors that are often inspired by biology (e.g. ants) [179]. Some examples in the literature use e-Puck or Kilobot swarms controlled by BTs created using evolutionary algorithms. In [138] a robot swarm is composed of twenty 2-wheeled e-puck robots, whose behavior is governed by BTs with a restricted topology that are automatically generated by the Maple planner. The hardware of the robots limits the pool of leaves to six actions and six conditions. The tasks executed by the swarm are *foraging* and *aggregation*. A similar swarm of sixteen e-pucks is used in [37] and controlled

by evolved BTs to push a frisbee in an arena. The same authors use evolved BTs to control a swarm of Kilobots to carry out a foraging task in [36]. Here, the BT approach is preferred because of its *transparency* to a human observer. Finally, in [38,39] BNF grammar implementing the Grammatical Evolution genotype-to-phenotype transformation incorporates rules that produce BTs to represent swarm behaviors. In this work, evolved behaviors are shown to perform better than a hand-coded one for a swarm performing single source foraging, nest maintenance and cooperative transportation. Robotic swarm control using BTs was also done in [115], as an execution module in a hierarchical framework.

Concerning walking robots, in [95,135], BTs are used to make the Softbank NAO robot walk towards and grasp a target object. The robot application is used as a proof of concept for the BT framework. In [110], BTs are used to adapt the walking capabilities of a simulated humanoid robot navigating through three different types of terrain. In this latter case, the *Fallback* node is exploited and it learns how to change the step depth of the robot depending on the terrain conditions. In [117], a hand-coded BT controls a Pepper robot for CRI (Child–Robot Interaction) "in the wild" experiments. A graphic block-based interface allows non-programmers to generate complex robot behaviors powered by BTs.

Finally, a research field in which BTs have not yet been applied to a large extent is Autonomous Driving. In most cases, as described in [180], the decision layer of an autonomous vehicle is a Finite State Machine (FSM) or Partially Observed Markov Decision Process (POMDP). Thus, given that BTs generalize these structures, an autonomous vehicle can also be controlled by a BT, and the advantages in terms of *modularity*, *reactivity* and *transparency* could be useful in unpredictable urban environments as well as highway scenarios. This is loosely suggested in [63], where a simple example includes a car performing lane following, overtaking and parking. These behaviors have also been simulated in [105] using both FSMs and BTs. It is shown how the latter scale better when compared to the former, when more behaviors are added, allowing increased reusability and structure simplicity.

### 3.2.3. Aerial and underwater robots

BTs have been used for two categories of Unmanned Aerial Vehicles (UAVs), small multi-copters moving in close proximity of obstacles [41,125] and large fixed wing aircraft moving at higher altitudes in longer missions [119,154].

For small UAVs, BTs were used in the winning design of the 2017 International Micro Aerial Vehicle Competition (IMAV 17) [125]. There, the UAV was tasked with moving through a window, search for an object, pass in front of a fan, find a location and drop an object there and finally land on a moving platform. The advantages of BTs in terms of *modularity* and hierarchical structure was mentioned as key elements of the approach used. A related task was addressed in [41] using evolutionary methods. Finally, manually designed BTs were used as a key part of a building inspection drone design in [164], and an LTL planning approach was used for a UAV search mission in [165]. In [128] an increase in automation was achieved by implementing BTs for UAV systems that operate in congested areas.

For larger UAVs, BTs were used to do mission management for extremely high altitude operations in a series of papers [119, 120,122,123], that include real, practical operations. BTs have also been proposed for the control of a UAV chasing other aircraft [154] as illustrated in Fig. 6.

Finally, BTs have been used to address the strict safety requirements that are associated with the operation of large UAVs. Issues regarding synchronization and memory were investigated to enable redundancy for BTs in [127], and BTs for fault detection and mitigation were proposed in [124].
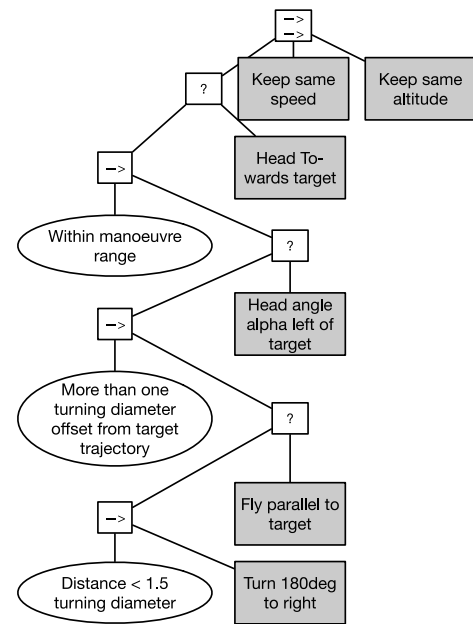


**Fig. 6.** A BT for UAVs to move towards and fly in behind another UAV. *Source:* Adapted from [154].

The operation of Autonomous Underwater Vehicles (AUVs) also requires very reliable systems, not to avoid damage to by-standers, but rather to reduce the risk of loosing the vehicle itself. Thus, a BT approach for AUV mission management, including fault handling, was proposed in [121].

### 3.3. Other

Behavior trees have also been used for a large variety of applications apart from those listed in previous sections (see Table 6).

Since a BT represents an action policy, they can be used as a general language for describing workflows and procedures to completely different types of tasks. One such class of tasks is found in medicine. Using BTs for online procedure guidance for emergency medical services has been proposed in [169], in a manner similar to dialog systems for games. Surgical procedures have also been formulated in BT notation, in an agent-agnostic way that allows the same description to be used for procedures to be carried out either manually by the surgeons themselves or by a robot [131]. Other examples have specifically targeted joint human–robot task execution. In one approach, human interaction was used to get input for a Fallback node, to take advantage of an experienced operator's (brain surgeon) expertise in choosing the correct action for a brain tumor ablation task. This has been demonstrated in both simulation [130] and in real experiments on mice [167].

Apart from using BTs to prescribe human tasks, they have also been used to describe observed actions and task executions. A conceptual model for how to represent general human workflows using BTs was presented in [129], along with a prototype demonstration of how to encode cooking procedures. In [166], activities of daily living have been observed as interaction constraints between tracked objects, which are then automatically converted into BT descriptions. Similar descriptions are used to encode activities for daily living in [168], where the demonstrated implementation is generating behaviors for virtual agents in a simulated home environment. Camera placement for animation

**Table 7**
Learning approaches to synthesizing a BT.

| Learning method | Papers |
|---|---|
| Reinforcement | [19,23,25,26,32,34,35,40,44, 133,181] |
| Evolutionary | [10–14,17,20–22,22,27–30, 33,36,37,37,38,38,39,43] |
| Case Based Reasoning | [35,46,50,78] |
| From Demonstration | [48,51,52] |



**Fig. 7.** A BT for Pacman, that is learned using evolutionary methods.
*Source:* Adapted from [16].

production in virtual environments can also be controlled with BTs [170,171].

Furthermore, the BT framework has been used to formalize procedures for problem solving. One example is to find closed form inverse kinematics for a general serial manipulator [176]. The procedure carried out by humans is encoded in a BT and the authors argue that the same approach can be applied to other complex but well-structured cognitive tasks. Another example is the design of power grid systems, where the design procedure is automated in a similar way using BTs [177].

Finally, apart from encoding task execution for autonomous agents and humans, BTs have been also been used to run simpler automated systems, such as smart homes. The *reactivity* of the BTs can be exploited to control a domestic power grid, by balancing loads and engaging local back-up power production when needed [173–175], or control the activation of lighting or smart appliances based on human actions [172].
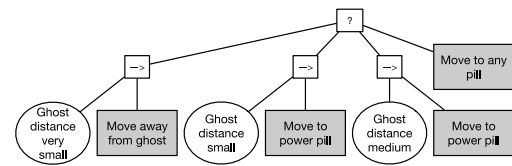
## 4. Methodology

BTs were originally conceived as a tool to manually create modular task switching structures. However, as *modularity* is beneficial for both manual and automatic synthesis, BTs have been successfully used as a policy representation for both learning and planning approaches, with the hope of improving both performance and design time.

### 4.1. Learning

The learning approaches to designing BTs can be divided into Reinforcement Learning, Evolution-inspired learning, Case Based Reasoning and Learning from demonstration, as seen in Table 7. Some of these methods have been applied to automatically generate BTs while other methods have been used to improve existing parameters within a predefined BT. Below we discuss different learning approaches in more detail.

#### 4.1.1. Reinforcement learning (RL)

In [19], an existing BT is re-ordered using Q-values that are computed using the lowest level Sequences as actions in the RL algorithm. In [23], the RL takes place in the Fallback nodes, executing the child with the highest Q-value. Similar ideas were built upon in [32] combining the hierarchical RL approach MAXQ with BTs. Starting from a manual design, some Fallback nodes are marked to be improved by learning. Having found the Q-values of each child, new condition nodes were created to allow execution of each child when that child is the best option. This line of ideas was also explored in [26]. The idea of combining RL and BTs was also discussed at a more general level in [133,181]. Individual actions or conditions can also be endowed with RL capabilities, a concept that was explored in [25,44]. The later specified learning nodes in the BT where an RL problem in terms of states, actions and rewards was defined. Finally, an approach involving the translation of a learned RL policy into a BT was suggested in [40].

#### 4.1.2. Evolution-inspired learning

Evolutionary Algorithms build upon the ideas of having a population of solution candidates divided into generations. For each generation, some kind of fitness evaluation is performed to select a subset of the generation, then a new generation is created from this subset by applying operations such as mutations and crossover. As seen below, BTs are very well suited to such algorithms. It has been shown that *locality*, in terms of small changes in design giving small changes in performance, is important for the performance of evolutionary algorithms [182]. Thus, it seems that the *modularity* of BTs is providing the locality needed by these algorithms.

Genetic Programming is one form of evolutionary algorithms that has been applied to BTs, see Fig. 7, and a number of studies have shown that solutions that outperform manual designs can be found [12–14,17,38]. These methods have been used to automatically evolve new behaviors using high-level definitions given a new environment [20,21] and additional improvements have been made to increase scalability for evolving algorithms [10]. A combination of Grammatical Evolution with BTs and path-finding algorithms such as A* was explored in [11]. Air Combat simulators are highly dynamical environments where human behavior is often represented by rule-based scripts. Grammatical Evolution has been used to create adaptive human behavior models [28] and studies have been done to improve pilot decision support during reconnaissance missions [30]. Other military-related training simulators included ground unit mobility and training on how to act in dangerous situations. To reduce the need for military experts to explain to programmers how virtual soldiers should behave in a particular situation, a genetic algorithm has been used to automatically generate such agents [29]. In multi-agent cases, it is often difficult to specify policies for every single agent and thus the same hand-written BTs have been used for many units. Using an evolutionary approach, individual BTs were developed for each agent, resulting in a more natural and efficient behavior of the whole team [39,43], but the downside of such automated techniques is reduced designer control [22]. Another example of a multi-agent system is a robotic swarm. In this field, BTs have been generated using both Grammatical Evolution [38] and Genetic Programming. In [36,37] the evolutionary system were based on a *distributed island model*. Some of the solutions generated by Evolutionary Algorithms can be quite large and complex. Both [17] and [27] describe methods for reducing this problem.

#### 4.1.3. Case based reasoning

Case Based Reasoning is an automated decision making process where solutions to new problems are provided through experiences from problems that were previously encountered. Following this process, rules can be constructed that may solve future problems by retrieving and reusing stored behaviors [46, 50]. One drawback of this method has been the difficulty of refining existing strategies that have been previously learned from experience [49], but it has been shown that some additional adaptability can be provided using RL [35].

**Table 8**
Different approaches for synthesizing a BT using a planning tool.

| Planner | Papers |
|---|---|
| STRIPS type | [134] |
| HTN | [133,136,139,183] |
| Auto-MoDe | [138] |
| LTL | [96,132,135,165] |
| GOAP | [22] |
| Other | [137,140–142] |

**Table 9**
Example of an input to the problem addressed in [134].

| Actions | Preconditions | Postconditions |
|---|---|---|
| $A_1$ | $C_{11}^{Pre}$, $C_{12}^{Pre}$ | $C^{Post}$ |
| $A_2$ | $C_{21}^{Pre}$, $C_{22}^{Pre}$ | $C^{Post}$ |

### 4.1.4. Learning from demonstration

Usually one needs an expert or a person with good knowledge in the field of robotics to program a robot for executing a particular task. Much less knowledge is required if the robot can learn from a situation where the human demonstrates a particular task that the robot needs to execute [52]. Similarly, creating an NPC for a game can be a time consuming process, where game designers have to interact with a programmer for creating the desired characters. Learning by demonstration has shown improvements in generating more human-like behavior, although in situations where human behavior was less predictable, the resulting NPC behaviors might not meet expectations [51]. Machine learning functions, such as neural networks, have been used in combination with BTs to let game developers combine hand-coded BTs with programming by demonstration [48]. This technique can increase performance in uncertain environments thanks to the noise tolerance of neural networks.

### 4.2. Planning and analytic design

In this section we review the papers that show how a BT, such as the one in Fig. 8, can be synthesized using a planner. Depending on the planner chosen, these papers can be categorized as in Table 8.

Planning algorithms are generally used to create action sequences to achieve long term goals. However, they have the important limitation that if the agent fails to execute an action or a subtask, it has to re-plan. Such failures are usually due to uncertainties. Perhaps the outcome of an action was different than expected, or the state of the world, in terms of positions and geometry of all sorts of objects and agents, was different than expected. Thus, the more uncertain, changing and dynamic the world is, the more often plans will fail. Also, re-planning takes time and computational resources, especially if the task is complex and involves many objects and actions.

BTs are inherently reactive, and by constantly checking conditions they switch tasks based on the perceived state of the world. However, the size of the BT must be very large to perform tasks that involve many objects and subtasks, and the larger the BT, the more difficult it is to design manually. The idea of using planning algorithms to construct BTs addresses this problem, trying to combine the *reactivity* of BTs with the goal directed nature of plans.

The most common approach to synthesize a BT from a planner consists of two steps: first, the planner computes a plan to solve a given task, then a planner-specific algorithm converts the plan into a BT which is finally used to control the agent (a robot in many cases).
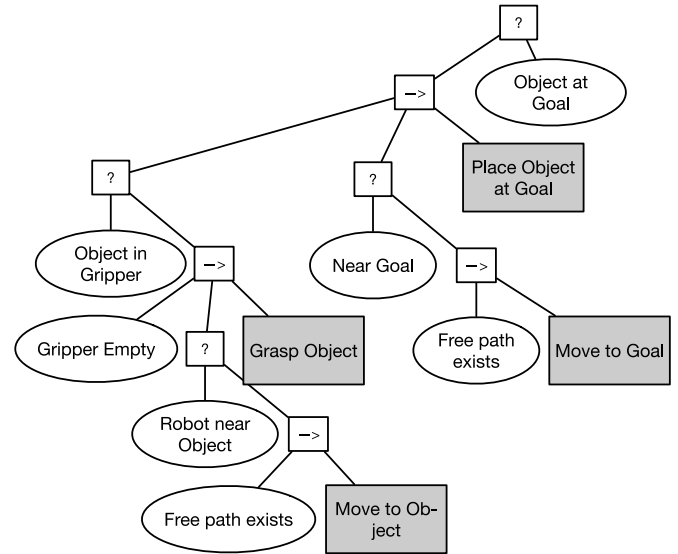
Several authors have proposed to generate BTs from a Hierarchical Task Network (HTN), as a natural extension of STRIPS-style planners, probably because this type of planning results in hierarchical task decompositions which have a natural mapping to the hierarchical structure of BTs.

In [134], the authors combine BTs, relying on their *reactivity* and *modularity*, with a STRIPS-style planning algorithm, leveraging the idea of backchaining: they start from the goal condition



**Fig. 8.** A BT for mobile manipulation, generated using a planning approach, and data from a table of actions and preconditions, as illustrated in Table 9.
*Source:* Adapted from [134].

and proceed backwards, iteratively finding the actions that meet that goal condition. Input to the problem is a set of actions with the corresponding preconditions and postconditions (Table 9). Furthermore, it is reactive to changes in the environment brought by external agents and it can be expanded during runtime.

Based on the same idea, [142] and [141] present their own algorithm for the automatic synthesis of BTs. This algorithm is combined with dynamic differential logic (DL), a verification tool used in [142] to ensure the safety of the generated BT. Then, DL is combined with a market-based auction algorithm and a coordinator to assign specific task to certain robots, in the multi-system scenario presented in [141]. In [137] a heuristic planner is used to create BTs in the form of Robust Logical-Dynamical Systems (RLDS).

In [183], a Game AI is built using HTN planners alongside BTs. The main contribution of this work is the proposition of an hybrid approach where HTN is used to compute high-level strategic plans while the BTs are managing low-level tasks. Despite the presence of the hybrid control strategy combining the planner with BTs, the BTs are still hand coded. An automatic transition from HTN to BTs is proposed in [136]. Here, the pre- and post-conditions of HTNs are added to BTs, in a new framework called eBT (extended BT). In particular, the HTN is initialized with a planner based on a PDDL (Planning Domain Definition Language) representation, translating a semantic model of the world, shared among all the nodes in the eBT. The overall procedure is accomplished in two phases: first the PDDL planner generates a sequence of skills (actions), then an eBT is generated and optimized for execution.

HTNs are combined with BTs and implemented in ROS in [139]. Here, the HTN planner computes a plan from input tasks, which

is automatically converted into a BT that executes it. Also, the user is free to define task-dependent subtrees which are automatically dealt with and combined with the pre-existing BTs. In this framework, the planning and execution layer communicate through a blackboard which is also responsible for tracking and checking the execution. If the actual current logic state does not correspond to the expected logic state, replanning is needed. The extended BT (XBT) proposed in [133] also uses HTN planning. It is proposed to create virtual copies of the states of the tree and evaluate it, in order to prune behaviors that either fail or do not contribute to reaching the goal.

Inspired by the *precondition/postcondition* mechanism in PDDL, [140] combine the BT representation model with the *Hierarchical task and motion Planning in the Now* (HPN) planning algorithm, to design an algorithm that synthesizes BTs automatically. The planner outputs a hierarchical sequence of post-conditions → actions → pre-conditions, which is then translated into a BT. At runtime, the algorithm also keeps track of the failing behaviors, thus allowing for replanning.

In [138] Maple, as an instance of AutoMoDe (automatic modular design) is used to automatically generate a restricted topology of BTs, limited to a Sequence* node[8] as a root, which has Fallback nodes as children, which in turn have one Condition and one Action leaf each. The number of subtrees rooted with the Fallback node is limited to four. Moreover, due to the robot setup, the BT leaves are chosen from a set of six actions and six conditions. Iterated F-race is used as an optimization algorithm to search for the best BT among the generated ones. The Maple method is compared with Chocolate (which, similarly, generates probabilistic FSMs) and EvoStick (which is an implementation of an evolutionary robotics approach that uses neural networks), in the tasks of *foraging* (retrieve an object from a source area and place it in a nest area), and *aggregation* (the swarm is asked to aggregate in an area), both in simulation and reality. In the first task Maple and Chocolate perform better than EvoStick both in simulation and reality. In the second task instead, EvoStick performs better in simulation but worse in reality when compared to Maple and Chocolate, which in turn have similar results.

An alternative approach to reactive planners is proposed in [135]. Here, the synthesis of a robot controller is a two-step process. In the first step, the action capabilities of the robot are modeled as a transition system and the goal as a State-Event LTL formula (Linear Temporal Logic). Then, an I/O automaton synthesizes the control strategy from the LTL formula, which is finally automatically implemented in a BT. In a similar scenario, [132] shows how to synthesize a BT that guarantees to satisfy a task defined as a LTL formula. The rigorous and complex formulations of the LTL formula and the I/O automaton make this approach scale poorly in more complicated tasks or less predictable environments. Moreover the BT is generated offline. LTL generated plans are also used in [165], where BTs refine high-level task plans expanding them into primitive actions.

Finally, [22] proposes a method that automatically generates a BT from GOAP (Goal-Oriented Action Planning). The method requires a symbolic definition of a problem, which can be solved by a GOAP algorithm, then computes the solution in three phases. First, a Monte-Carlo simulation outputs a set of solutions produced by the algorithm for different initial states. Then, the most representative sequences in this solution set are identified by building a N-Gram model from the Monte-Carlo simulation. Finally, the N-Gram model is merged through a genetic algorithm into a BT that mimics the execution of the GOAP algorithm.

---

[8] *Sequence* with memory.

**Table 10**
Different aspects of manual design.

| Type of design | Papers |
|---|---|
| Design principles | [9,102,134,162] |
| Design tools | [4,55,57,98–101,105,117,151,152,155] |
| Hybrid approaches | [42,44,97,122,183] |

### 4.3. Manual design and other approaches

The majority of the BTs used for the applications described in Section 3 are designed manually. The reason for this is that BTs were created to support manual design, and the methods for synthesizing BTs automatically using learning and planning are not yet mature enough to compete with the ease of manual design, especially in cases where the required BT is fairly small. Therefore, alongside the development on automatic synthesis, there has also been work on how to support the manual design process, as seen in Table 10.

Many solutions suggest hybrid approaches. In [110] the BTs are hand-coded but the Fallback node is endowed with the capability of learning from experience. In [44], specific action nodes have learning capabilities (Q-learning in particular). In [183] hand coded BTs are used alongside the HTN planner. In [42], a BT is manually designed to include a Neural Network Controller. In [122] BTs are manually coded to translate a formalism from the ALC(D) description logic, and in [97], BTs are manually combined with Motion Generators.

Some authors have created and used supporting tools to sketch and design the BTs. In [57] the sketch-based authoring tool *AIPaint* is presented. The same year, [55] introduced the graphical interface *Topiary*. *RIZE* is another graphic interface, based on building blocks, which is used in [117] to design BTs. [105] constructed their own plugin BT editor for *Unity3D*, and today there are also many others available. Lastly, the *CoSTAR* user interface has been developed over a number of publications [98–101].

In [151] and [152] an Active BT method is used, which is implemented in ABL (A Behavior Language), a reactive planning language used for Game AI. One of the benefits of this language is claimed to be the scheduling of the actions, which is handled by a planner. Note that one still has to hand-code the desired behaviors for the AI agent according to the ABL semantics, even though the transition to the ABT is automatic. ABL is used also in [4,153].

When doing manual design, inspiration can be drawn from the fact that BTs have been shown to generalize a number of other control architectures, [102]. Such design principles are discussed in detail in [9]. For some problems, the choice of action is naturally done in a Decision Tree fashion, checking conditions on different levels to finally end up in a suitable leaf of the tree. Note that Decision Trees have no return status, so it is a special case of general BT design. For other problems, a so-called implicit sequence design, inspired by the Teleo-Reactive approach [162] might be better, or a backward chained approach as suggested in [134].

Another use of manually designed BTs is proposed in [176], where a hand-coded BT is used to solve the Inverse Kinematics of a robot, up to 6 DoF, taking as input the Denavit–Hartenberg parameters. The outputs of this application are Python and C++ code as well as a report of the results in LaTex.

## 5. Implementation

In this section we give an overview of some of the most common libraries available to implement BTs. We also list the implementation language, the presence of a GUI, whether they

**Table 11**
Some of the BT libraries available on November 2019.

| Name | Language | GUI | ROS | Open source | Forks | Stars | Last commit |
|---|---|---|---|---|---|---|---|
| py_trees[a] | Python | | ✓ | ✓ | 36 | 87 | *up to date* |
| Owyl[b] | Python | | | ✓ | 13 | 60 | Nov. 29, 2014 |
| Playful[c] | Python | ✓ | | ✓ | 0 | 3 | Nov. 1, 2019 |
| behave[d] | Python | | | ✓ | 9 | 45 | May 11, 2015 |
| task_behavior_engine[e] | Python | ✓ | | ✓ | 9 | 27 | Aug 3, 2018 |
| gdxAI[f] | Java | | | ✓ | 189 | 758 | Jul. 26, 2019 |
| bte2[g] | Java | | ✓ | ✓ | 5 | 27 | Aug 6, 2018 |
| Behavior3[h] | JavaScript | ✓ | | ✓ | 135 | 335 | Oct. 21, 2018 |
| BehaviorTree.js[i] | JavaScript | | | ✓ | 29 | 207 | May 3, 2019 |
| NPBehave[j] | Unity3D/C# | ✓ | | ✓ | 52 | 306 | Oct. 24, 2019 |
| BT-Framework[k] | Unity3D/C# | | | ✓ | 66 | 163 | Mar 2, 2015 |
| fluid-behavior-tree[l] | Unity3D/C# | | | ✓ | 15 | 107 | Jun 13, 2019 |
| hivemind[m] | Unity3D/C# | ✓ | | ✓ | 13 | 47 | May 14, 2015 |
| BehaviorTree.CPP[n] | C++ | ✓ | ✓ | ✓ | 77 | 306 | *up to date* |
| ROS-Behavior-Tree[o] | C++ | ✓ | ✓ | ✓ | 48 | 154 | Oct. 22, 2018 |
| Behavior Tree Starter Kit[p] | C++ | | | ✓ | 103 | 297 | Jun 24, 2014 |
| BrainTree[q] | C++ | | | ✓ | 27 | 121 | Aug 23, 2018 |
| Behavior-Tree[r] | C++ | ✓ | | ✓ | 33 | 116 | Oct 17, 2018 |

[a] https://github.com/splintered-reality/py_trees.
[b] https://github.com/eykd/owyl.
[c] https://playful.is.tuebingen.mpg.de/.
[d] https://github.com/fuchen/behave.
[e] https://github.com/ToyotaResearchInstitute/task_behavior_engine.
[f] https://github.com/libgdx/gdx-ai.
[g] https://github.com/piotr-j/bte2.
[h] https://github.com/behavior3.
[i] https://github.com/Calamari/BehaviorTree.js.
[j] https://github.com/meniku/NPBehave.
[k] https://github.com/f15gdsy/BT-Framework.
[l] https://github.com/ashblue/fluid-behavior-tree.
[m] https://github.com/rev087/hivemind.
[n] https://github.com/BehaviorTree/BehaviorTree.CPP.
[o] https://github.com/miccol/ROS-Behavior-Tree.
[p] https://github.com/aigamedev/btsk.
[q] https://github.com/arvidsson/BrainTree.
[r] https://github.com/miccol/Behavior-Tree.

can communicate with ROS, and whether they are open source. Moreover, we report some data from the Github repository to give the reader a sense of the usage of the library. Please note that this list is not complete, as there is a very large set of alternatives out there.

py_trees (extended to py_trees_ros[9]) is one of the most used BT libraries, possibly because it is open source and well maintained. The core library is aimed at game AI design, but it has extensions for robotic implementations in ROS. It features the following design constraints[10]:

- no interaction or sharing of data between tree instances;
- no parallelization of tree execution;
- only one behavior initializing or executing at a time.

Visualizing the BT is also possible thanks to ASCII or dot graph output. Information sharing between nodes is enabled by a *blackboard*, which allow for sharing information within nodes in the same BT but not between different BTs. A notable limitation of this library is that the *Sequence* node is implemented "with memory", thus removing one of the advantages of using a BT, i.e. *reactivity*. [9] notes that "the use of nodes with memory is advised exclusively for those cases where there is no unexpected event that will undo the execution of the subtree in a composition with memory". To overcome this limitation, each user needs to implement a memoryless version of the Sequence node, which is quite straightforward using Fallbacks and negations. The *modularity* of BTs allows to program a specific sub-tree and then take it

out-of-the-shelf when needed; however, the use of the *blackboard* is inherently task specific, thus limiting *modularity*.

Owyl is an early BT library and one of the first implemented in Python. To our knowledge it has not been widely used nor is still maintained.

gdxAI is a framework written in Java for game development with libGDX. It supports features for AI agents movement, pathfinding and decision making, both via BTs and FSMs.

Behavior3 is a library developed by the authors of [44]. It features a visual editor, namely Behavior3 Editor (corresponding to the data in Table 11), which can export the modeled trees to JSON files, and to a client library for Python and Javascript. Even though a visual editor simplifies the design and the readability, an extensive documentation, as well as periodic maintenance are lacking. ROS support is also not implemented.

BehaviorTree.js is another library addressed to the design of game AI implemented in Javascript. Note that the return state of the nodes of this library can be only *Success* or *Failure*, no *Running*.

NPBehave is a BT library that targets game AI for the Unreal Engine as are BT-Framework, fluid-behavior-tree and hivemind, which comes with a visual editor with runtime visual debugging capabilities. In particular, NPBehave is an event driven library, i.e., it features event driven behavior trees which do not need to be executed from the root node at each tick. This design is claimed to be more efficient and simpler to use.

ROS-Behavior-Tree is a library implemented by the authors in [63]. According to the documentation, the latest version of ROS is not currently supported and the last commit is dated October

---

9   https://github.com/splintered-reality/py_trees_ros
10  from https://py-trees.readthedocs.io/en/devel/background.html

2018. A probable reason for this state is because the authors collaborated in the creation of `BehaviorTree.CPP` which features the Groot GUI Editor and an implementation in ROS.

## 6. Open challenges

In this section we point out four important open challenges for BT research: *Explainable AI, Human–robot-interaction, safe AI*, and *the combination of learning and BTs*.

Explainable AI is an area of increasing importance, especially in the light of the recent success of data driven methods such as Reinforcement Learning and Deep Learning. BTs are widely believed to provide good explainability, as they have been used to capture human workflows [129,131], and are in general well suited for manual design. Furthermore, Hierarchical Task Networks (HTNs) is a planning tool built on the intuitive idea of high level tasks being composed of low level tasks in several layers, and the parallels between BTs and HTNs have been explored in [183]. Finally, human readability is a well known advantage of BTs [36,101,169]. Together, these ideas indicate that BTs can have a role to play in explainable AI, either by incorporating data driven methods into BTs, or by learning a BT from the workings of a data driven policy.

Human–robot-interaction (HRI) and collaboration is another area that will increase in importance as robots are more integrated in factories and homes. Explainable AI is certainly a part of HRI, but areas such as programming by non experts, and human in the loop collaborations are also important. Again, the human readability of BTs, together with *modularity*, will simplify these issues. Important progress has already been made in [99–101], but much remains to be done.

Safe autonomy is a third area related to explainable AI and HRI. To be able to guarantee safety of a robot system will be increasingly important as autonomous systems that are big enough to harm humans, such as unmanned vehicles, start to share spaces with people. The *transparency* and *modularity* of BTs can play an important role in this area. Some initial work in this area was done in [42], but many questions remain.

Finally, as seen in Section 4.1, the combination of BTs and ML is being explored by several research groups. It is clear that end-to-end RL will not provide a viable solution to many problems, as in some cases the state space is just to large to explore efficiently, and in others, lack of explainability and/or safety guarantees makes such approaches infeasible. However, both these problems could be addressed by a BT solution where individual nodes, leaves as well as interior ones, are improved by learning methods.

## 7. Conclusions

In this survey paper we have provided an overview of the over 160 research papers devoted to the development of BT as a tool for AI in games and robotics. We have partitioned and analyzed the papers based on both application areas and methods used, and finally provided a description of a number of open challenges to the research community.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Matteo Iovino and Jonathan Styrud are employed by ABB Corporate Research (SE) and ABB Robotics (SE) respectively, while Edvards Scukins is employed by SAAB Aeronautics (SE).

## Acknowledgments

## References

[1] R. Dromey, From requirements to design: Formalizing the key steps, in: First International Conference OnSoftware Engineering and Formal Methods, 2003. Proceedings, IEEE, Brisbane, Queensland, Australia, 2003, pp. 2–11, http://dx.doi.org/10.1109/SEFM.2003.1236202.

[2] B. Iske, U. Ruckert, A methodology for behaviour design of autonomous systems, in: Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180), 1, IEEE, 2001, pp. 539–544.

[3] R. Marcotte, H.J. Hamilton, Behavior trees for modelling artificial intelligence in games: A tutorial, The Computer Games Journal 6 (3) (2017) 171–184, http://dx.doi.org/10.1007/s40869-017-0040-9.

[4] M. Mateas, A. Stern, A behavior language for story-based believable agents, IEEE Intell. Syst. 17 (4) (2002) 39–47, http://dx.doi.org/10.1109/MIS.2002.1024751.

[5] D. Isla, Handling complexity in the halo 2 AI, in: Gamasutra - GDC 2005 Proceeding, 2005, https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php?print=1.

[6] G. Florez-Puga, M.A. Gomez-Martin, P.P. Gomez-Martin, B. Diaz-Agudo, P.A. Gonzalez-Calero, Query-enabled behavior trees, IEEE Transactions on Computational Intelligence and AI in Games 1 (4) (2009) 298–308, http://dx.doi.org/10.1109/TCIAIG.2009.2036369.

[7] P. Ögren, Increasing modularity of UAV control systems using computer game behavior trees, in: AIAA Guidance, Navigation, and Control Conference 2012; Minneapolis, MN; United States; 13 August 2012 Through 16 August 2012, 2012.

[8] J.A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T.Y. Liu, N. Pollard, M. Pivtoraiko, J. Valois, R. Zhu, An integrated system for autonomous robotics manipulation, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp. 2955–2962, http://dx.doi.org/10.1109/IROS.2012.6385888.

[9] M. Colledanchise, P. Ögren, Behavior Trees in Robotics and AI : an Introduction, CRC Press, 2018, http://dx.doi.org/10.1201/9780429489105, KTH.

[10] Q. Zhang, K. Xu, P. Jiao, Q. Yin, Behavior modeling for autonomous agents based on modified evolving behavior trees, in: 2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS), 2018, pp. 1140–1145, http://dx.doi.org/10.1109/DDCLS.2018.8515939.

[11] M. Nicolau, D. Perez-Liebana, M. O'Neill, A. Brabazon, Evolutionary behavior tree approaches for navigating platform games, IEEE Trans. Comput. Intell. AI in Games 9 (3) (2017) 227–238, http://dx.doi.org/10.1109/TCIAIG.2016.2543661.

[12] C.-U. Lim, R. Baumgarten, S. Colton, Evolving behaviour trees for the commercial game DEFCON, in: D. Hutchison, T. Kanade, J. Kittler, J.M. Kleinberg, F. Mattern, J.C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M.Y. Vardi, G. Weikum, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A.I. Esparcia-Alcazar, C.-K. Goh, J.J. Merelo, F. Neri, M. Preuß, J. Togelius, G.N. Yannakakis (Eds.), Applications of Evolutionary Computation, vol. 6024, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 100–110, http://dx.doi.org/10.1007/978-3-642-12239-2_11.

[13] D. Perez, M. Nicolau, M. O'Neill, A. Brabazon, Evolving behaviour trees for the Mario AI competition using grammatical evolution, in: C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A.I. Esparcia-Alcázar, J.J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, G.N. Yannakakis (Eds.), Applications of Evolutionary Computation, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 123–132.

[14] J.W. Hoff, H.J. Christensen, Evolving behaviour trees: - automatic generation of ai opponents for real-time strategy games, (Ph.D. thesis), NTNU, 2016.

[15] L. Peña, S. Ossowski, J.M. Peña, S.M. Lucas, Learning and evolving combat game controllers, in: 2012 IEEE Conference on Computational Intelligence and Games (CIG), 2012, pp. 195–202, http://dx.doi.org/10.1109/CIG.2012.6374156.

[16] Q. Zhang, J. Yao, Q. Yin, Y. Zha, Learning behavior trees for autonomous agents with hybrid constraints evolution, Appl. Sci. 8 (7) (2018) 1077, http://dx.doi.org/10.3390/app8071077.

[17] M. Colledanchise, R. Parasuraman, P. Ögren, Learning of behavior trees for autonomous agents, IEEE Trans. Games 11 (2) (2019) 183–189, http://dx.doi.org/10.1109/TG.2018.2816806.

[18] B.J. Oakes, Practical and theoretical issues of evolving behaviour trees for a turn-based game, (Ph.D. thesis), McGill University, 2013.

[19] R. Dey, C. Child, QL-BT: Enhancing behaviour tree design and implementation with Q-learning, in: 2013 IEEE Conference on Computational Inteligence in Games (CIG), IEEE, Niagara Falls, ON, Canada, 2013, pp. 1–8, http://dx.doi.org/10.1109/CIG.2013.6633623.

[20] M. Estgren, E.S.V. Jansson, Behaviour tree evolution by genetic programming, Linköping University, 2017.

[21] C. Paduraru, M. Paduraru, Automatic difficulty management and testing in games using a framework based on behavior trees and genetic algorithms, in: ArXiv:1909.04368 [Cs], 2019.

[22] P. Schwab, H. Hlavacs, Capturing the essence: Towards the automated generation of transparent behavior models, in: Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, 2015.

[23] Y. Fu, L. Qin, Q. Yin, A reinforcement learning behavior tree framework for game AI, in: Proceedings of the 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering, Atlantis Press, Huhhot, China, 2016, http://dx.doi.org/10.2991/essaeme-16.2016.120.

[24] C.-U. Lim, An A.I. player for DEFCON: An evolutionary approach using behavior trees, (Ph.D. thesis), Imperial College of London, 2009.

[25] M. Kartasev, Integrating reinforcement learning into behavior trees by hierarchical composition, (Ph.D. thesis), KTH Royal Institute of Technology, 2019.

[26] X. Zhu, Behavior tree design of intelligent behavior of non-player character (NPC) based on Unity3D, J. Intell. Fuzzy Systems 37 (5) (2019) 6071–6079, http://dx.doi.org/10.3233/JIFS-179190.

[27] A. Hallawa, S. Schug, G. Iacca, G. Ascheid, Evolving instinctive behaviour in resource-constrained autonomous agents using grammatical evolution, in: P.A. Castillo, J.L. Jiménez Laredo, F. Fernández de Vega (Eds.), Applications of Evolutionary Computation, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 369–383, http://dx.doi.org/10.1007/978-3-030-43722-0_24.

[28] J. Yao, Q. Huang, W. Wang, Adaptive human behavior modeling for air combat simulation, in: 2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2015, pp. 100–103, http://dx.doi.org/10.1109/DS-RT.2015.12.

[29] G. Berthling-Hansen, E. Morch, R.A. Løvlid, O.E. Gundersen, Automating behaviour tree generation for simulating troop movements (poster), in: 2018 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA), 2018, pp. 147–153, http://dx.doi.org/10.1109/COGSIMA.2018.8423978.

[30] P. Eilert, Learning behaviour trees for simulated fighter pilots in airborne reconnaissance missions : A grammatical evolution approach, (Ph.D. thesis), Linköping University, 2019.

[31] J. Yao, Q. Huang, W. Wang, Adaptive CGFs based on grammatical evolution, Math. Probl. Eng. 2015 (2015) http://dx.doi.org/10.1155/2015/197306.

[32] Q. Zhang, L. Sun, P. Jiao, Q. Yin, Combining behavior trees with MAXQ learning to facilitate CGFs behavior modeling, in: 2017 4th International Conference on Systems and Informatics (ICSAI), 2017, pp. 525–531, http://dx.doi.org/10.1109/ICSAI.2017.8248348.

[33] J. Yao, W. Wang, Z. Li, Y. Lei, Q. Li, Tactics exploration framework based on genetic programming, Int. J. Comput. Intell. Syst. 10 (1) (2017) 804–814, http://dx.doi.org/10.2991/ijcis.2017.10.1.53.

[34] Q. Zhang, Q. Yin, Y. Hu, Modeling CGFs behavior by an extended option based learning behavior trees, in: 2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM), 2017, pp. 260–265, http://dx.doi.org/10.1109/ICCIS.2017.8274784.

[35] Q. Zhang, Q. Yin, K. Xu, Towards an integrated learning framework for behavior modeling of adaptive CGFs, in: 2016 9th International Symposium on Computational Intelligence and Design (ISCID), 2, 2016, pp. 7–12, http://dx.doi.org/10.1109/ISCID.2016.2011.

[36] S. Jones, M. Studley, S. Hauert, A. Winfield, Evolving behaviour trees for swarm robotics, in: R. Groß, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, M. Gauci (Eds.), Distributed Autonomous Robotic Systems: the 13th International Symposium, in: Springer Proceedings in Advanced Robotics, Springer International Publishing, Cham, 2018, pp. 487–501, http://dx.doi.org/10.1007/978-3-319-73008-0_34.

[37] S. Jones, M. Studley, S. Hauert, A. Winfield, A two teraflop swarm, Front. Robot AI 5 (FEB) (2018) http://dx.doi.org/10.3389/frobt.2018.00011.

[38] A. Neupane, Emergence of collective behaviors in hub-based colonies using grammatical evolution and behavior trees, (Ph.D. thesis), Brigham Young University, 2019.

[39] A. Neupane, M. Goodrich, Designing emergent swarm behaviors using behavior trees and grammatical evolution, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, 4, 2019, pp. 2138–2140.

[40] B. Banerjee, Autonomous acquisition of behavior trees for robot control, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 3460–3467, http://dx.doi.org/10.1109/IROS.2018.8594083.

[41] K.Y.W. Scheper, S. Tijmons, C.C. de Visser, G.C.H.E. de Croon, Behavior trees for evolutionary robotics, Artif. Life 22 (1) (2015) 23–48, http://dx.doi.org/10.1162/ARTL_a_00192.

[42] C.I. Sprague, P. Ögren, Adding neural network controllers to behavior trees without destroying performance guarantees, ArXiv:1809.10283 [Cs] (2018).

[43] A. Neupane, M. Goodrich, Learning swarm behaviors using grammatical evolution and behavior trees, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Macao, China, 2019, pp. 513–520, http://dx.doi.org/10.24963/ijcai.2019/73.

[44] R.d.P. Pereira, P.M. Engel, A framework for constrained and adaptive behavior-based agents, ArXiv:1506.02312 [Cs] (2015).

[45] G. Robertson, I. Watson, Building behavior trees from observations in real-time strategy games, in: 2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA), 2015, pp. 1–7, http://dx.doi.org/10.1109/INISTA.2015.7276774.

[46] R. Palma, P.A. González-Calero, M.A. Gómez-Martín, P.P. Gómez-Martín, Extending case-based planning with behavior trees, in: Twenty-Fourth International FLAIRS Conference, 2011.

[47] I. Sagredo-Olivenza, P.P. Gómez-Martín, M.A. Gómez-Martín, P.A. González-Calero, Trained behavior trees: Programming by demonstration to support AI game designers, IEEE Trans. Games 11 (1) (2019) 5–14, http://dx.doi.org/10.1109/TG.2017.2771831.

[48] I. Sagredo-Olivenza, P.P. Gómez-Martín, M.A. Gómez-Martín, P.A. González-Calero, Combining neural networks for controlling non-player characters in games, in: I. Rojas, G. Joya, A. Catala (Eds.), Advances in Computational Intelligence, in: Lecture Notes in Computer Science, Springer International Publishing, 2017, pp. 694–705.

[49] R. Palma, A.A. Sánchez-Ruiz, M.A. Gómez-Martín, P.P. Gómez-Martín, P.A. González-Calero, Combining expert knowledge and learning from demonstration in real-time strategy games, in: A. Ram, N. Wiratunga (Eds.), Case-Based Reasoning Research and Development, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 181–195.

[50] G. Flórez-Puga, M. Gómez-Martín, B. Díaz-Agudo, P.A. González-Calero, Dynamic expansion of behaviour trees, in: Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, in: AIIDE'08, AAAI Press, Stanford, California, 2008, pp. 36–41.

[51] C. Buche, C. Even, J. Soler, Orion: A generic model and tool for data mining, in: M.L. Gavrilova, C.K. Tan, A. Sourin (Eds.), Transactions on Computational Science XXXVI: Special Issue on Cyberworlds and Cybersecurity, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2020, pp. 1–25, http://dx.doi.org/10.1007/978-3-662-61364-1_1.

[52] K. French, S. Wu, T. Pan, Z. Zhou, O.C. Jenkins, Learning behavior trees from demonstration, in: 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 7791–7797, http://dx.doi.org/10.1109/ICRA.2019.8794104.

[53] M. Cutumisu, D. Szafron, An architecture for game behavior AI: Behavior multi-queues, in: Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, in: AIIDE'09, AAAI Press, Stanford, California, 2009, pp. 20–27.

[54] I. Millington, J. Funge, Artificial Intelligence for Games, CRC Press, 2009.

[55] A. Shoulson, F.M. Garcia, M. Jones, R. Mead, N.I. Badler, Parameterizing behavior trees, in: J.M. Allbeck, P. Faloutsos (Eds.), Motion in Games, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 144–155.

[56] I. Bojic, T. Lipic, M. Kusek, G. Jezic, Extending the JADE agent behaviour model with JBehaviourTrees framework, in: J. O'Shea, N.T. Nguyen, K. Crockett, R.J. Howlett, L.C. Jain (Eds.), Agent and Multi-Agent Systems: Technologies and Applications, vol. 6682, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 159–168, http://dx.doi.org/10.1007/978-3-642-22000-5_18.

[57] D. Becroft, J. Bassett, A. Mejia, C. Rich, C. Sidner, AIPaint: A sketch-based behavior tree authoring tool, in: Seventh Artificial Intelligence and Interactive Digital Entertainment Conference, 2011.

[58] A. Johansson, P. Dell'Acqua, Emotional behavior trees, in: 2012 IEEE Conference on Computational Intelligence and Games (CIG), 2012, pp. 355–362, http://dx.doi.org/10.1109/CIG.2012.6374177.

[59] S. Ocio, Adapting AI behaviors to players in Driver San Francisco: Hinted-execution behavior trees, in: Eighth Artificial Intelligence and Interactive Digital Entertainment Conference, 2012.

[60] B. Merrill, Building utility decisions into your existing behavior tree, in: Game AI Pro, CRC Press, 2013, pp. 127–136.

[61] A. Shoulson, M.L. Gilbert, M. Kapadia, N.I. Badler, An event-centric planning approach for dynamic real-time narrative, in: Proceedings of Motion on Games - MIG '13, ACM Press, Dublin 2, Ireland, 2013, pp. 121–130, http://dx.doi.org/10.1145/2522628.2522629.

[62] M. Colledanchise, A. Marzinotto, P. Ögren, Performance analysis of stochastic behavior trees, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 3265–3272, http://dx.doi.org/10.1109/ICRA.2014.6907328.

[63] M. Colledanchise, P. Ögren, How behavior trees modularize robustness and safety in hybrid systems, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014, pp. 1482–1488, http://dx.doi.org/10.1109/IROS.2014.6942752.

[64] T. Plch, M. Marko, P. Ondracek, M. Cerny, J. Gemrot, C. Brom, An AI system for large open virtual world, in: Tenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2014.

[65] C. Dominguez, Y. Ichimura, M. Kapadia, Automated interactive narrative synthesis using dramatic theory, in: Proceedings of the 8th ACM SIG-GRAPH Conference on Motion in Games, MIG 2015, 2015, pp. 103–112, http://dx.doi.org/10.1145/2822013.2822028.

[66] W.P. Subagyo, S.M.S. Nugroho, S. Sumpeno, Simulation multi behavior NPCs in fire evacuation using emotional behavior tree, in: 2016 International Seminar on Application for Technology of Information and Communication (ISemantic), 2016, pp. 184–190, http://dx.doi.org/10.1109/ISEMANTIC.2016.7873835.

[67] Y. Li, D.-W. Xu, A game AI based on ID3 algorithm, in: 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), IEEE, Greater Noida, India, 2016, pp. 681–687, http://dx.doi.org/10.1109/IC3I.2016.7918049.

[68] M. Waltham, D. Moodley, An analysis of artificial intelligence techniques in multiplayer online battle arena game environments, in: Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists, in: SAICSIT '16, ACM, Johannesburg, South Africa, 2016, pp. 45:1–45:7, http://dx.doi.org/10.1145/2987491.2987513.

[69] S. Rabin, Game AI Pro 3 : Collected Wisdom of Game AI Professionals, A K Peters/CRC Press, 2017, http://dx.doi.org/10.4324/9781315151700.

[70] B. Dagerman, High-level decision making in adversarial environments using behavior trees and Monte Carlo tree search, (Ph.D. thesis), KTH Royal Institute of Technology, 2017.

[71] R.A. Agis, S. Gottifredi, A.J. García, An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games, Expert Syst. Appl. (2020) 113457, http://dx.doi.org/10.1016/j.eswa.2020.113457.

[72] L. Sun, A. Shoulson, P. Huang, N. Nelson, W. Qin, A. Nenkova, N.I. Badler, Animating synthetic dyadic conversations with variations based on context and agent attributes, Comput. Animat. Virtual Worlds 23 (1) (2012) 17–32, http://dx.doi.org/10.1002/cav.1421.

[73] F. Geraci, M. Kapadia, Authoring background character responses to foreground characters, in: International Conference on Interactive Digital Storytelling ICIDS 2015, 2015, pp. 130–141.

[74] L. Li, G. Liu, M. Zhang, Z. Pan, E. Song, BAAP: A behavioral animation authoring platform for emotion driven 3D virtual characters, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6243 LNCS, 2010, pp. 350–357.

[75] L.A. Ripamonti, S. Gratani, D. Maggiorini, D. Gadia, A. Bujari, Believable group behaviours for NPCs in FPS games, in: 2017 IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 12–17, http://dx.doi.org/10.1109/ISCC.2017.8024497.

[76] A. Johansson, P. Dell'Acqua, Comparing behavior trees and emotional behavior networks for NPCs, in: 2012 17th International Conference on Computer Games (CGAMES), 2012, pp. 253–260, http://dx.doi.org/10.1109/CGames.2012.6314584.

[77] M. Kapadia, J. Falk, F. Zünd, M. Marti, R. Sumner, M. Gross, Computer-assisted authoring of interactive narratives, in: Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, I3D 2015, 2015, pp. 85–92.

[78] G. Flórez-Puga, D. Llansó, M. Gómez-Martín, P. Gómez-Martín, B. Díaz-Agudo, P. González-Calero, Empowering designers with libraries of self-validated query-enabled behaviour trees, in: Artificial Intelligence for Computer Games, Springer, 2011, pp. 55–81, http://dx.doi.org/10.1007/978-1-4419-8188-2_3.

[79] T. Triebel, M. Lehn, R. Rehner, B. Guthier, S. Kopf, W. Effelsberg, Generation of synthetic workloads for multiplayer online gaming benchmarks, in: 2012 11th Annual Workshop on Network and Systems Support for Games (NetGames), 2012, pp. 1–6, http://dx.doi.org/10.1109/NetGames.2012.6404028.

[80] M.N.M. Othman, H. Haron, Implementing game artificial intelligence to decision making of agents in emergency egress, in: 2014 8th. Malaysian Software Engineering Conference (MySEC), 2014, pp. 316–320, http://dx.doi.org/10.1109/MySec.2014.6986036.

[81] Y. Wang, L. Wang, J. Liu, Object behavior simulation based on behavior tree and multi-agent model, in: 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2017, pp. 833–836, http://dx.doi.org/10.1109/ITNEC.2017.8284851.

[82] S. Delmer, Behavior trees for hierarchical RTS AI, (Ph.D. thesis), Southern Methodist University, Plano, Tx, USA, 2012.

[83] C. Martens, O. Iqbal, Villanelle: An authoring tool for autonomous characters in interactive fiction, in: R.E. Cardona-Rivera, A. Sullivan, R.M. Young (Eds.), Interactive Storytelling, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2019, pp. 290–303, http://dx.doi.org/10.1007/978-3-030-33894-7_29.

[84] C. Buche, C. Even, J. Soler, Autonomous virtual player in a video game imitating human players: The ORION framework, in: 2018 International Conference on Cyberworlds (CW), 2018, pp. 108–113, http://dx.doi.org/10.1109/CW.2018.00029.

[85] I. Sagredo-Olivenza, M.A. Gómez-Martín, P.A. González-Calero, Supporting the collaboration between programmers and designers building game AI, in: K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, R. Malaka (Eds.), Entertainment Computing - ICEC 2015, in: Lecture Notes in Computer Science, Springer International Publishing, 2015, pp. 496–501.

[86] P.A. González Calero, M.A. Gómez-Martín (Eds.), Artificial intelligence for computer games, Springer, New York, 2011.

[87] E. Tomai, R. Salazar, R. Flores, Simulating aggregate player behavior with learning behavior trees, in: 22nd Annual Conference on Behavior Representation in Modeling and Simulation, BRiMS 2013 - Co-Located with the International Conference on Cognitive Modeling, 2013, pp. 85–92.

[88] G. Brown, D. King, Facilitating open plot structures in story driven video games using situation generation, in: 13th International Conference on Intelligent Games and Simulation, GAME-on 2012, 2012, pp. 34–38.

[89] S. Ocio, Building a risk-free environment to enhance prototyping, in: Game AI Pro 2, CRC Press, 2015.

[90] J. Gong, H. Wang, Q. Liu, J. Huang, J. Hao, Agent behavior combination modeling by multi-method integration, in: Proceedings of the 5th International Conference on Frontiers of Educational Technologies, in: ICFET 2019, ACM, Beijing, China, 2019, pp. 137–142, http://dx.doi.org/10.1145/3338188.3338214.

[91] G. Xiaobing, L. Sijiang, J. Yongshi, Y. Yiping, A visual authoring framework for intelligent flight trainer, in: 2010 International Conference on Computer and Information Application, 2010, pp. 382–385, http://dx.doi.org/10.1109/ICCIA.2010.6141616.

[92] X. Geng, S. Qin, H. Chang, Y. Yang, A hybrid knowledge representation for the domain model of intelligent flight trainer, in: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, 2011, pp. 29–33, http://dx.doi.org/10.1109/CCIS.2011.6045026.

[93] A. Woolley, J. Liu, Cognitive modelling of crew response during naval damage control, in: RINA, Royal Institution of Naval Architects - International Conference on Human Factors in Ship Design and Operation, Papers, 2016-September, 2016.

[94] A.J. Kamrud, D.D. Hodson, G.L. Peterson, B.G. Woolley, Unified behavior framework in discrete event simulation systems, The J. Def Model Simul 14 (4) (2017) 471–481, http://dx.doi.org/10.1177/1548512916683450.

[95] A. Marzinotto, M. Colledanchise, C. Smith, P. Ögren, Towards a unified behavior trees framework for robot control, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 5420–5427, http://dx.doi.org/10.1109/ICRA.2014.6907656, KTH.

[96] A. Marzinotto, Flexible robot to object interactions through rigid and deformable cages, (Ph.D. thesis), KTH Royal Institute of Technology, 2017.

[97] F. Rovida, D. Wuthier, B. Grossmann, M. Fumagalli, V. Krüger, Motion generators combined with behavior trees: A novel approach to skill modelling, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 5964–5971, http://dx.doi.org/10.1109/IROS.2018.8594319.

[98] K.R. Guerin, C. Lea, C. Paxton, G.D. Hager, A framework for end-user instruction of a robot assistant for manufacturing, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 6167–6174, http://dx.doi.org/10.1109/ICRA.2015.7140065.

[99] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, G.D. Hager, CoSTAR: Instructing collaborative robots with behavior trees and vision, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 564–571, http://dx.doi.org/10.1109/ICRA.2017.7989070.

[100] C. Paxton, F. Jonathan, A. Hundt, B. Mutlu, G.D. Hager, User experience of the CoSTAR System for instruction of collaborative robots, 2017, ArXiv:1703.07890 [Cs].

[101] C. Paxton, F. Jonathan, A. Hundt, B. Mutlu, G.D. Hager, Evaluating methods for end-user creation of robot task plans, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 6086–6092, http://dx.doi.org/10.1109/IROS.2018.8594127.

[102] M. Colledanchise, P. Ögren, How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees, IEEE Trans. Robot. 33 (2) (2017) 372–389, http://dx.doi.org/10.1109/TRO.2016.2633567, KTH.

[103] M. Colledanchise, L. Natale, Analysis and exploitation of synchronized parallel executions in behavior trees, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019.

[104] A. Csiszar, M. Hoppe, S.A. Khader, A. Verl, Behavior trees for task-level programming of industrial robots, in: T. Schüppstuhl, J. Franke, K. Tracht (Eds.), Tagungsband Des 2. Kongresses Montage Handhabung Industrieroboter, Springer, Berlin, Heidelberg, 2017, pp. 175–186, http://dx.doi.org/10.1007/978-3-662-54441-9_18.

[105] M. Olsson, Behavior trees for decision-making in autonomous driving, (Ph.D. thesis), KTH Royal Institute of Technology, 2016.

[106] M. Colledanchise, A. Marzinotto, D.V. Dimarogonas, P. Ögren, The advantages of using behavior trees in mult-robot systems, in: Proceedings of ISR 2016: 47st International Symposium on Robotics, 2016, pp. 1–8.

[107] M. Colledanchise, L. Natale, Improving the parallel execution of behavior trees, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 7103–7110, http://dx.doi.org/10.1109/IROS.2018.8593504.

[108] Y. Jiang, N. Walker, M. Kim, N. Brissonneau, D.S. Brown, J.W. Hart, S. Niekum, L. Sentis, P. Stone, LAAIR: A layered architecture for autonomous interactive robots, 2018, ArXiv:1811.03563 [Cs].

[109] M. Kim, M. Arduengo, N. Walker, Y. Jiang, J.W. Hart, P. Stone, L. Sentis, An architecture for person-following using active target search, 2018, ArXiv:1809.08793 [Cs].

[110] B. Hannaford, D. Hu, D. Zhang, Y. Li, Simulation results on selector adaptation in behavior trees, 2016, ArXiv:1606.09219 [Cs].

[111] R.H. Abiyev, N. Akkaya, E. Aytac, Control of soccer robots using behaviour trees, in: 2013 9th Asian Control Conference (ASCC), 2013, pp. 1–6, http://dx.doi.org/10.1109/ASCC.2013.6606326.

[112] R.H. Abiyev, I. Günsel, N. Akkaya, E. Aytac, A. Çağman, S. Abizada, Robot soccer control using behaviour trees and fuzzy logic, 12th International Conference on Application of Fuzzy Systems and Soft Computing, ICAFS 2016, 29-30 August 2016, Vienna, Austria, Procedia Comput. Sci. 102 (2016) 477–484, http://dx.doi.org/10.1016/j.procs.2016.09.430.

[113] F.D.L. Siqueira, E.R.D. Pieri, A context-aware approach to the navigation of mobile robots, in: XII Simposio Brasileiro de Automacao Inteligente (SBAI), 2015, http://dx.doi.org/10.13140/RG.2.1.3715.8481.

[114] F.d.L. Siqueira, P.D.M. Plentz, E.R.D. Pieri, Semantic trajectory applied to the navigation of autonomous mobile robots, in: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), 2016, pp. 1–8, http://dx.doi.org/10.1109/AICCSA.2016.7945766.

[115] Q. Yang, Z. Luo, W. Song, R. Parasuraman, Hierarchical needs based framework for reactive multi-robot planning with dynamic tasks, 2019.

[116] E. Giunchiglia, M. Colledanchise, L. Natale, A. Tacchella, Conditional behavior trees: Definition, executability, and applications, in: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), 2019, pp. 1899–1906, http://dx.doi.org/10.1109/SMC.2019.8914358.

[117] E. Coronado, X. Indurkhya, G. Venture, Robots meet children, development of semi-autonomous control systems for children-robot interaction in the wild, in: 2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM), 2019, pp. 360–365, http://dx.doi.org/10.1109/ICARM.2019.8833751.

[118] S. Macenski, F. Martín, R. White, J.G. Clavero, The Marathon 2: A navigation system, 2020, ArXiv:2003.00368 [Cs].

[119] A. Klöckner, Behavior trees for UAV mission management, in: IN-FORMATIK 2013: Informatik Angepasst An Mensch, Organisation Und Umwelt, P-220, 2013, pp. 57–68.

[120] A. Klöckner, Behavior trees with stateful tasks, in: J. Bordeneuve-Guibé, A. Drouin, C. Roos (Eds.), Advances in Aerospace Guidance, Navigation and Control, Springer International Publishing, Cham, 2015, pp. 509–519, http://dx.doi.org/10.1007/978-3-319-17518-8_29.

[121] C.I. Sprague, O. Özkahraman, A. Munafo, R. Marlow, A. Phillips, P. Ögren, Improving the modularity of AUV control systems using behaviour trees, in: 2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV), 2018, pp. 1–6, http://dx.doi.org/10.1109/AUV.2018.8729810.

[122] A. Klöckner, Interfacing behavior trees with the world using description logic, in: AIAA Guidance, Navigation, and Control (GNC) Conference, American Institute of Aeronautics and Astronautics, 2018, http://dx.doi.org/10.2514/6.2013-4636.

[123] A. Klöckner, The modelica BehaviorTrees library: Mission planning in continuous-time for unmanned aircraft, in: The 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden, 2014, pp. 727–736, http://dx.doi.org/10.3384/ecp14096727.

[124] L. Castano, H. Xu, Safe decision making for risk mitigation of UAS, in: 2019 International Conference on Unmanned Aircraft Systems (ICUAS), 2019, pp. 1326–1335, http://dx.doi.org/10.1109/ICUAS.2019.8797774.

[125] M. Lan, Y. Xu, S. Lai, B.M. Chen, A modular mission management system for micro aerial vehicles, in: 2018 IEEE 14th International Conference on Control and Automation (ICCA), 2018, pp. 293–299, http://dx.doi.org/10.1109/ICCA.2018.8444251.

[126] V. Golluecke, D. Lange, A. Hahn, S. Schweigert, Behavior tree based knowledge reasoning for intelligent vessels in maritime traffic simulations, in: ECMS 2018 Proceedings Edited By Lars Nolle, Alexandra Burger, Christoph Tholen, Jens Werner, Jens Wellhausen, ECMS, 2018, pp. 105–113, http://dx.doi.org/10.7148/2018-0105.

[127] E. Safronov, M. Vilzmann, D. Tsetserukou, K. Kondak, Asynchronous behavior trees with memory aimed at aerial vehicles with redundancy in flight controller, ArXiv:1907.00253 [Cs] (2019).

[128] H. Goudarzi, D. Hine, A. Richards, Mission automation for drone inspection in congested environments, in: 2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS), 2019, pp. 305–314, http://dx.doi.org/10.1109/REDUAS47371.2019.8999719.

[129] W. Deneke, L. Xu, C. Thompson, A conceptual model of human workflows, in: 2017 Second International Conference on Information Systems Engineering (ICISE), 2017, pp. 54–58, http://dx.doi.org/10.1109/ICISE.2017.8.

[130] D. Hu, Y. Gong, B. Hannaford, E.J. Seibel, Semi-autonomous simulated brain tumor ablation with RAVENII surgical robot using behavior tree, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 3868–3875, http://dx.doi.org/10.1109/ICRA.2015.7139738.

[131] B. Hannaford, R. Bly, I. Humphreys, M. Whipple, Behavior trees as a representation for medical procedures, 2018, ArXiv:1808.08954 [Cs].

[132] M. Colledanchise, R.M. Murray, P. Ögren, Synthesis of correct-by-construction behavior trees, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 6039–6046, http://dx.doi.org/10.1109/IROS.2017.8206502.

[133] M. Hölzl, T. Gabor, Reasoning and learning for awareness and adaptation, in: M. Wirsing, M. Hölzl, N. Koch, P. Mayer (Eds.), Software Engineering for Collective Autonomic Systems: the ASCENS Approach, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2015, pp. 249–290, http://dx.doi.org/10.1007/978-3-319-16310-9_7.

[134] M. Colledanchise, D. Almeida, P. Ögren, Towards blended reactive planning and acting using behavior trees, in: IEEE International Conference on Robotics and Automation, IEEE, Montreal, Canada, 2019, KTH.

[135] J. Tumova, A. Marzinotto, D.V. Dimarogonas, D. Kragic, Maximally satisfying LTL action planning, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, Chicago, IL, USA, 2014, pp. 1503–1510, http://dx.doi.org/10.1109/IROS.2014.6942755.

[136] F. Rovida, B. Grossmann, V. Krüger, Extended behavior trees for quick definition of flexible robotic tasks, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 6793–6800, http://dx.doi.org/10.1109/IROS.2017.8206598.

[137] C. Paxton, N. Ratliff, C. Eppner, D. Fox, Representing robot task plans as robust logical-dynamical systems, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 5588–5595.

[138] J. Kuckling, A. Ligot, D. Bozhinoski, M. Birattari, Behavior trees as a control architecture in the automatic modular design of robot swarms, in: M. Dorigo, M. Birattari, C. Blum, A.L. Christensen, A. Reina, V. Trianni (Eds.), Swarm Intelligence, in: Lecture Notes in Computer Science, Springer International Publishing, 2018, pp. 30–43.

[139] J.A. Segura-Muros, J. Fernández-Olivares, Integration of an automated hierarchical task planner in ROS Using Behaviour Trees, in: 2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT), 2017, pp. 20–25, http://dx.doi.org/10.1109/SMC-IT.2017.11.

[140] H. Zhou, H. Min, Y. Lin, An autonomous task algorithm based on behavior trees for robot, in: 2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI), 2019, pp. 64–70, http://dx.doi.org/10.1109/CCHI.2019.8901959.

[141] T.G. Tadewos, L. Shamgah, A. Karimoddini, On-the-fly decentralized tasking of autonomous vehicles, in: 2019 IEEE 58th Conference on Decision and Control (CDC), 2019, pp. 2770–2775, http://dx.doi.org/10.1109/CDC40024.2019.9029554.

[142] T.G. Tadewos, L. Shamgah, A. Karimoddini, Automatic safe behaviour tree synthesis for autonomous agents, in: 2019 IEEE 58th Conference on Decision and Control (CDC), 2019, pp. 2776–2781, http://dx.doi.org/10.1109/CDC40024.2019.9030183.

[143] D. Llansó, M. Gómez-Martín, P. González-Calero, Self-validated behaviour trees through reflective components, in: Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2009, 2009, pp. 70–75.

[144] Y.A. Sekhavat, Behavior trees for computer games, Int. J. Artif. Intell. Tools 26 (02) (2017) 1730001, http://dx.doi.org/10.1142/S0218213017300010.

[145] T. Plch, M. Marko, P. Ondráček, M. Černý, J. Gemrot, C. Brom, Modular behavior trees: Language for fast AI in open-world video games, in: ECAI, 2014, http://dx.doi.org/10.3233/978-1-61499-419-0-1209.

[146] L.X. Ji, J.H. Ma, Research on the behavior of intelligent role in computer games based on behavior tree, Appl. Mech. Mater. 509 (2014) 165–169, http://dx.doi.org/10.4028/www.scientific.net/AMM.509.165.

[147] J. Tremblay, Understanding and evaluating behaviour trees, McGill University, 2012.

[148] J.T. Balint, J.M. Allbeck, R. Bidarra, Understanding everything NPCs can do: Metrics for action similarity in non-player characters, in: Proceedings of the 13th International Conference on the Foundations of Digital Games, in: FDG '18, ACM, New York, NY, USA, 2018, pp. 14:1–14:10, http://dx.doi.org/10.1145/3235765.3235776.

[149] X. Zhang, D. Schaumann, B. Haworth, P. Faloutsos, M. Kapadia, Coupling agent motivations and spatial behaviors for authoring multiagent narratives, Comput. Animat. Virtual Worlds 30 (3–4) (2019) e1898, http://dx.doi.org/10.1002/cav.1898.

[150] S. Belle, C. Gittens, T.C.N. Graham, Programming with affect: How behaviour trees and a lightweight cognitive architecture enable the development of non-player characters with emotions, in: 2019 IEEE Games, Entertainment, Media Conference (GEM), 2019, pp. 1–8, http://dx.doi.org/10.1109/GEM.2019.8811542.

[151] B.G. Weber, P. Mawhorter, M. Mateas, A. Jhala, Reactive planning idioms for multi-scale game AI, in: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, IEEE, Copenhagen, Denmark, 2010, pp. 115–122, http://dx.doi.org/10.1109/ITW.2010.5593363.

[152] B.G. Weber, M. Mateas, A. Jhala, Building human-level AI for real-time strategy games, in: 2011 AAAI Fall Symposium Series, 2011.

[153] C. Simpkins, S. Bhat, C. Isbell, M. Mateas, Towards adaptive programming, in: Proceedings of the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications, 2008, p. 11.

[154] M. Ramirez, L. Benke, M. Papasimeon, N. Lipovetzky, A. Pearce, M. Zamani, E. Scala, T. Miller, Integrated hybrid planning and programmed control for real-time UAV maneuvering, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, 2, 2018, pp. 1318–1326.

[155] V. Berenz, S. Schaal, The playful software platform: Reactive programming for orchestrating robotic behavior, IEEE Robot. Autom. Mag. 25 (3) (2018) 49–60, http://dx.doi.org/10.1109/MRA.2018.2803168.

[156] E. Safronov, Node templates to improve reusability and modularity of behavior trees, ArXiv:2002.03167 [Cs] (2020).

[157] A.J. Champandard, P. Dunstan, P. Dunstan, The behavior tree starter kit, in: S. Rabin (Ed.), Game AI Pro, First, CRC Press, 2013, pp. 27–46, http://dx.doi.org/10.1201/9780429055058-3.

[158] M. Colledanchise, P. Ögren, How behavior trees generalize the teleo-reactive paradigm and and-or-trees, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, pp. 424–429, http://dx.doi.org/10.1109/IROS.2016.7759089.

[159] J. Chen, D. Shi, Development and composition of robot architecture in dynamic environment, in: Proceedings of the 2018 International Conference on Robotics, Control and Automation Engineering, in: RCAE 2018, ACM, Beijing, China, 2018, pp. 96–101, http://dx.doi.org/10.1145/3303714.3303716.

[160] B. Hannaford, Hidden Markov models derived from behavior trees, 2019, ArXiv:1907.10029 [Cs].

[161] E.W. Dijkstra, Go to statement considered harmful, Commun. ACM 11 (3) (1968) 147–148.

[162] N. Nilsson, Teleo-reactive programs for agent control, J. Artificial Intelligence Res. 1 (1993) 139–158.

[163] E. Coronado, F. Mastrogiovanni, G. Venture, Development of intelligent behaviors for social robots via user-friendly and modular programming tools, in: 2018 IEEE Workshop on Advanced Robotics and Its Social Impacts (ARSO), 2018, pp. 62–68, http://dx.doi.org/10.1109/ARSO.2018.8625839.

[164] G.-Y. Li, R.-T. Soong, J.-S. Liu, Y.-T. Huang, UAV System integration of real-time sensing and flight task control for autonomous building inspection task, in: 2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI), 2019, pp. 1–6, http://dx.doi.org/10.1109/TAAI48200.2019.8959825.

[165] M. Lan, S. Lai, T.H. Lee, B.M. Chen, Autonomous task planning and acting for micro aerial vehicles, in: 2019 IEEE 15th International Conference on Control and Automation (ICCA), 2019, pp. 738–745, http://dx.doi.org/10.1109/ICCA.2019.8899502.

[166] F. Bergeron, S. Giroux, K. Bouchard, S. Gaboury, RFID based activities of daily living recognition, in: 2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2017, pp. 1–5, http://dx.doi.org/10.1109/UIC-ATC.2017.8397548.

[167] D. Hu, Y. Gong, E.J. Seibel, L.N. Sekhar, B. Hannaford, Semi-autonomous image-guided brain tumour resection using an integrated robotic system: A bench-top study, The Int. J. Med Robot Comput Assist Surgery 14 (1) (2018) e1872, http://dx.doi.org/10.1002/rcs.1872.

[168] B. Bouchard, K. Bouchard, S. Gaboury, Y. Francillette, Modeling human activities using behaviour trees in smart homes, in: ACM International Conference Proceeding Series, 2018, pp. 67–74, http://dx.doi.org/10.1145/3197768.3201522.

[169] S. Shu, S. Preum, H.M. Pitchford, R.D. Williams, J. Stankovic, H. Alemzadeh, A behavior tree cognitive assistant system for emergency medical services, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2019, p. 8.

[170] D. Prima, B. Ferial Java, E. Suryapto, M. Hariadi, Secondary camera placement in machinema using behavior trees, in: 2013 International Conference on Quality in Research, QiR 2013 - in Conjunction with ICCS 2013: the 2nd International Conference on Civic Space, 2013, pp. 94–98, http://dx.doi.org/10.1109/QiR.2013.6632544.

[171] D. Markowitz, J.T. Kider, A. Shoulson, N.I. Badler, Intelligent camera control using behavior trees, in: J.M. Allbeck, P. Faloutsos (Eds.), Motion in Games, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 156–167.

[172] Y. Francillette, S. Gaboury, A. Bouzouane, B. Bouchard, Towards an adaptation model for smart homes, in: 14th International Conference on Smart Homes and Health Telematics, ICOST 2016, 9677, 2016, pp. 83–94, http://dx.doi.org/10.1007/978-3-319-39601-9_8.

[173] G. Brusco, A. Burgio, L. Mendicino, D. Menniti, M. Motta, A. Pinnarelli, N. Sorrentino, A compact nanogrid with a behavior-tree control for islanded applications and remote areas, in: 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), 2018, pp. 1–6, http://dx.doi.org/10.1109/ICNSC.2018.8361298.

[174] A. Burgio, D. Menniti, N. Sorrentino, A. Pinnarelli, M. Motta, A compact nanogrid for home applications with a behaviour-tree-based central controller, Appl. Energy 225 (2018) 14–26, http://dx.doi.org/10.1016/j.apenergy.2018.04.082.

[175] X. Haijun, Z. Qi, Simulation and real time analysis of network protection tripping strategy based on behavior trees, Cluster Comput. 22 (3) (2019) 5269–5278, http://dx.doi.org/10.1007/s10586-017-1207-7.

[176] D. Zhang, B. Hannaford, IKBT: Solving symbolic inverse kinematics with behavior tree, J. Artificial Intelligence Res. 65 (2019) 457–486, http://dx.doi.org/10.1613/jair.1.11592.

[177] A. Parfenyuk, Development of technology of through planning of electric power distribution systems based on consistent informative model, in: 2015 16th International Conference on Computational Problems of Electrical Engineering (CPEE), 2015, pp. 139–141, http://dx.doi.org/10.1109/CPEE.2015.7333359.

[178] S. Karakovskiy, J. Togelius, The mario ai benchmark and competitions, IEEE Trans. Comput Intell AI in Games 4 (1) (2012) 55–67.

[179] L.E. Parker, Multiple mobile robot systems, in: Handbook of Robotics, Springer, 2008, pp. 921–941.

[180] B. Paden, M. Čáp, S.Z. Yong, D. Yershov, E. Frazzoli, A survey of motion planning and control techniques for self-driving urban vehicles, IEEE Trans. Intell Veh 1 (1) (2016) 33–55, http://dx.doi.org/10.1109/TIV.2016.2578706.

[181] M. Hölzl, T. Gabor, Continuous collaboration: A case study on the development of an adaptive cyber-physical system, in: 2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems, 2015, pp. 19–25, http://dx.doi.org/10.1109/SEsCPS.2015.12.

[182] F. Rothlauf, M. Oetzel, On the locality of grammatical evolution, in: European Conference on Genetic Programming, Springer, 2006, pp. 320–330.

[183] X. Neufeld, S. Mostaghim, S. Brand, A hybrid approach to planning and execution in dynamic environments through hierarchical task networks and behavior trees, in: Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2018.

**Matteo Iovino** received a M.Sc. in Automation Engineering in 2018 at the University of Padua (IT) and a M.Sc. in General Engineering, with a specialization in Robotics, in 2018 at the École Centrale de Nantes (FR). He is currently employed by ABB Corporate Research (SE), while carrying out a Ph.D. in Computer Science at the Division of Robotics, Perception and Learning at the Royal Institute of Technology in Stockholm (KTH). His research interests include control and planning for collaborative mobile manipulation in dynamic industrial environments, with a focus on using learning techniques to synthesize control policies.

**Edvards Scukins** obtained his M.Sc. in aerospace engineering from Luleå University of Technology (Sweden) in 2016. He is working at the department of Aeronautical Solutions at SAAB Aeronautics and is currently pursuing his industrial Ph.D. in collaboration with the Division of Robotics, Perception and Learning at Royal Institute of Technology (Sweden). His research interests include the implementation of behavior trees for autonomous agents combined with reinforcement learning approaches to increase pilot safety in adversarial environments.

**Petter Ögren** was born in Stockholm, Sweden, in 1974. He received the M.S. degree in engineering physics and the Ph.D. degree in applied mathematics from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 1998 and 2003, respectively. In the fall of 2001, he visited the Mechanical Engineering Department, Princeton University, Princeton, NJ. From 2003 to 2012 he worked as a senior scientist and deputy research director in Autonomous Systems at the Swedish Defence Research Agency (FOI). In 2013 he started as an Associate Professor at KTH, and since 2018, he is a Professor at the division of Robotics, Perception and Learning (RPL) at KTH.

**Jonathan Styrud** received his M.Sc. in Engineering Physics in 2011 at Uppsala University. He is currently employed at ABB Robotics while also pursuing a Ph.D. in Computer Science at the Royal Institute of Technology in Stockholm (KTH). His research interests include control, motion planning and modeling for manipulation for collaborative industrial robotics. He is especially interested in using learning techniques to generate transparent control policies.

**Christian Smith** received his M.Sc. in Engineering Physics in 2005, and Ph.D. in Computer Science in 2009, both at the Royal Institute of Technology in Stockholm (KTH). He was a post-doctoral researcher at Advanced Telecommunications Research International (ATR) in Kyoto, Japan from 2010 to 2011. He is currently an Associate Professor in Computer Science and the Deputy Head of the Division of Robotics, Perception, and Learning at KTH. He is the secretary of the IEEE Robotics and Automation Society Swedish Chapter, and serves as associate editor for IEEE Robotics and Automation Letters, and several conference editorial boards. His research interests include control and modeling for manipulation and grasping in human-centric environments, such as medical facilities and small-scale manufacturing plants, as well as physical human–robot interaction.