Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

A Hybrid Approach to Planning and Execution in Dynamic Environments Through Hierarchical Task Networks and Behavior Trees

Xenija Neufeld
Faculty of Computer Science
Otto von Guericke University
Magdeburg, Germany,
Crytek GmbH, Frankfurt, Germany

Sanaz Mostaghim
Faculty of Computer Science
Otto von Guericke University
Magdeburg, Germany

Sandy Brand
Crytek GmbH
Frankfurt, Germany

Abstract

Intelligent autonomous agents that are acting in dynamic environments in real-time are often required to follow long-term strategies while also remaining reactive and being able to act deliberately. In order to create intelligent behaviors for video game characters, there are two common approaches – planners are used for long-term strategical planning, whereas Behavior Trees allow for reactive acting. Although both methodologies have their advantages, when used on their own, they fail to fully achieve both requirements described above. In this work, we propose a hybrid approach combining a Hierarchical Task Network planner for high-level planning while delegating low-level decision making and acting to Behavior Trees. Furthermore, we compare this approach with a pure planner in a multi-agent environment.

1 Introduction

Following long-term strategies while acting deliberately is an important requirement for autonomous agents that are acting in real-time in dynamic environments. Such environments can be found in the area of robotics, simulations or video games. There are two common groups of approaches that are used for the creation of intelligent behaviors of video game characters – planners and Behavior Trees.

Planners such as STRIPS (*Stanford Research Institute Problem Solver*) (Fikes and Nilsson 1971), a modified version of it – GOAP (*Goal Oriented Action Planner*) (Orkin 2006) – or the HTN (*Hierarchical Task Network*) planner (Ghallab, Nau, and Traverso 2004, Chapter 11.5) allow for strategical decision making and long-term planning. They are used in multiple commercial video games (Neufeld et al. 2017) as well as in some academic game research environments and benchmarks (Menif, Jacopin, and Cazenave 2014; Vassos and Papakonstantinou ; Soemers and Winands 2016). On the other hand, Behavior Trees are still used in the majority of video games as richer extensions of Finite State Machines (Rabin 2009). Constantly monitoring the environment, they allow for reactive decision making and acting.

However, both types of approaches have their disadvantages and, when used on their own, fail to stay reactive while following a long-term goal. Planners are good at the strategic level but do not incorporate reactive behavior. If a created plan fails during execution (for example due to unexpected changes in the environment), a new plan needs to be created. In games, such changes are very likely to happen due to unpredictable player behavior. This usually leads to a high re-planning frequency and only short plans (Jacopin 2014; Nau and Champandard 2012). Behavior Trees, in contrast, can easily adapt the agent's behavior to changes. However, they do not take previous or future game states into consideration and thus do not allow for long-term strategies.

Staying reactive while following a high-level goal becomes an even bigger problem if coordination of multiple agents that follow a joint goal is required. As already pointed out in Palma et al. (2011), especially when a group of agents is controlled through a planner on a strategical level and single agents follow their individual plans on a tactical level without adjusting their actions to unexpected changes, one of the following problems will arise: either 1) a low-level action will continue to be executed as long as its plan preconditions hold, even if some events in the world suggests that the action should be changed or 2) if the action's preconditions become invalidated, the whole plan fails and a new plan needs to be created even though the higher levels of the plan might still be valid.

In this work, we propose a hybrid approach combining an HTN planner (which is responsible for the coordination of multiple agents) with Behavior Trees. We assume that the lower levels of an HTN – which represent actions of single agents – are more likely to fail due to small changes in the environment, however the overall plan might still be valid and most likely require only minimal changes. For example, in a game, the preconditions for a coordinated group attack might still be valid but the path of a single agent might be blocked. Taking an alternative path for the agent would not invalidate the group plan. However, when using a pure planner approach, re-planning would be required at this point in order to find a new solution to the problem.

Since such failures can only be recognized during execution, we propose using Behavior Trees in order to observe the environment, make decisions at run-time and execute lower-level tasks of an HTN. In this case, the planner creates a high-level plan, without going into details of how the lower-level tasks will be executed. Using preconditions and effects of higher hierarchy levels (described in section 3.1), the planner assumes that the low-level tasks will succeed

during execution. In order to achieve a flawless execution of the overall plan, the responsible Behavior Trees are created in such a manner, that they achieve the effects of the plan task that they represent under the given preconditions. Only the failure of all Behavior Tree branches is propagated as a task failure to the planner triggering a re-planning.

The main contribution of this work is a comparison of the robustness of HTN plans that are executed without run-time decision making and adaptation to environmental changes, and the novel hybrid approach which is able to adapt low-level plan steps. The rest of the paper is structured as follows: in section 2, we describe the most relevant work so far. Section 3 summarizes the two approaches used in this work: HTN planners and Behavior Trees. Afterwards, we describe the proposed hybrid solution in section 4 which follows by a description of our experimental setup and and its results in section 5. Finally, we conclude the paper with some outline for future work in section 6.

2 Related Work

Creating behaviors for agents in dynamic environments is a very complex task. Dini et al. describe important aspects which are to be considered when designing a robust planning and execution system for virtual worlds (Dini et al. 2006). They consider a) plan recovery as a major motivation for putting a monitoring system on top of a planner. Further problems are b) uncertainty, c) dynamics of the environments, d) distributed plans and teamwork, e) maintenance of the environment data, f) authorability of the planning domain and g) constraints on the plan content (Dini et al. 2006). They describe *contingency planning* (Dearden et al. 2003) as a possible approach to deal with a broken plan if there is a way to anticipate where a plan might fail. In that sense, our approach is similar to contingency planning. The major difference is, however, that in contingency planning a *plan* for *every* possible contingency is created *offline*. In our case, possible solutions to sub-tasks (*parts of the plan*) are encoded as Behavior Trees which can be re-used across multiple situations and the decision on which actions to execute is made *at runtime* resulting in a more flexible behavior and a lower usage of runtime memory.

Two works that are most related to our approach are described in Weber et al. (2010, 2011) and Palma et al. (2011). In both approaches, case-based planning is used to select high-level goals in a real-time strategy (RTS) game and Behavior Trees are used for low-level decision-making. The case bases contain samples of recorded games, played by experts. In Weber et al. (2010, 2011), after selecting a goal from the case base, multiple *manager systems* make high-level decisions such as 'which buildings to build next' or 'where to place them'. Afterwards, a so-called *Active Behavior Tree* is *created at run-time*. In Palma et al. (2011), it is possible that multiple Behavior Trees represent the same kind of a tactical action. The selection of a particular Behavior Tree is then done *measuring the similarity* of the current game state and the states provided by each of the trees.

While inspired by the idea of using a planner on higher levels and Behavior Trees on the lower levels, there are differences between these two approaches and our work. First, we argue that using an HTN planner instead of a case base might provide better authorability and maintainability of the system which is very important for complex game environments. Second, as pointed out in Palma et al. (2011), the major disadvantage of case-based planning lies in the size of the case base that is required in order to cover all possible situations. A planning domain that is hand-crafted by a designer with expert knowledge of the domain might be more flexible and extensible. Third, instead of creating *one* shared Behavior Tree *for all* agents at runtime (as in Weber et al. (2010, 2011)), we use *one* of the predefined Behavior Trees *for each* agent according to the high-level task (similar to Palma et al. (2011)). This allows us to re-use predefined trees and maintain them in accordance to the plan tasks. Furthermore, single agents can independently switch between Behavior Trees while proceeding with their own plans without affecting each other's behavior.

Another approach is used in the area of robotics (Colledanchise, Almeida, and Ögren 2016). Here, the Hybrid Backward-Forward (HBF) algorithm (Garrett, Lozano-Pérez, and Kaelbling 2015) is used to create reachability graphs with possible action sequences (plans) that lead to a given goal which are then used to *create* Behavior Trees. Even though this approach allows for reactive behavior while following a high-level plan, it might create very large and unmaintainable Behavior Trees for long-term goals. In contrast to the graphs created by the HBF algorithm, the hierarchical structure of an HTN is more intuitive and similar to human reasoning (see section 3.1). Furthermore, because of their hierarchical structure, HTNs allow for distribution of tasks and coordination between multiple agents.

Another work that combines elements of Behavior Trees and planners is described in Hilburn (2014). Here, a planner makes uses of typical Behavior Tree elements such as *Selector* or *Sequence Nodes* simulating possible plan outcomes. Although this approach leads to an extended decision making in the *planning phase*, it is unclear how it impacts the *execution* of the created plans.

Further works that address the topic of (multi-agent) planning under uncertainty include *epistemic planning* (Engesser et al. 2015; Muise et al. 2015; Bolander 2017) which focuses on the notion of *beliefs* about the world state. The planner tries to *reason* about parts of the world state that are not known to it using an epistemic model of the world. Our approach is related to epistemic planning in the sense that it replaces epistemic notions that are created during the planning phase by decision making through Behavior Trees during the execution.

3 Background

3.1 HTN Planner

A Hierarchical Task Network planner generates plans by decomposing a high-level *compound* task that needs to be accomplished by an agent into a sequence of further compound or *primitive* sub-tasks. Thus, the tasks build a hierarchical network. The *divide-and-conquer*-decomposition of tasks resembles much more human reasoning than classical

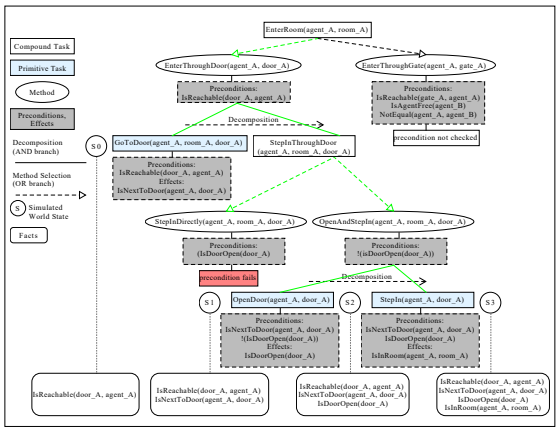


Figure 1: HTN for the *Enter a Room* example.

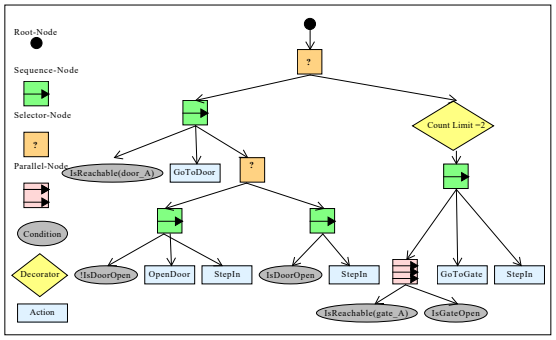


Figure 2: Behavior Tree for the *Enter a Room* example.

planning techniques, where a plan is created by searching a space of world states.

A compound task might be decomposed in multiple different ways through different *methods*. For example in order to *enter a room*, an agent might use a door or a gate as shown in Figure 1. In order to select a decomposition method, the planner checks the *preconditions* of a compound task's methods and applies a valid method. For example, the method *EnterThroughDoor* requires the door to be reachable by the agent. Entering through a door decomposes into the primitive task *GoToDoor* and another compound task *StepInThroughDoor*. In a depth-first manner, the decomposition process continues until all compound tasks are decomposed. The resulting plan is a sequence of primitive tasks. If a decomposition fails, the plan created by this decomposition is reverted, the failure is back-propagated and the next method is evaluated.

In order to be able to plan further into the future, the planner needs to reason about changes in the world that are caused by its plan steps. Therefore, it holds an inner representation of the world state which is described by so-called *facts*. For example, the fact *IsDoorOpen(door_A)* in Figure 1 is true if the door instance *door_A* is open. Changes in the world state are described by the *effects* of primitive tasks which add or delete facts to or from the planner's inner world representation. Similarly to methods, primitive tasks have *preconditions* under which they are applicable and which are true if corresponding facts exist in the planner's current world state. For example, the primitive task *StepIn(agent_A, door_A)* in our example is only applicable after the task *OpenDoor(agent_A, door_A)* adds the effect *IsDoorOpen(door_A)* to the simulated world state *S2*.

3.2 Behavior Trees

Since the representation of AI characters in the game *Halo 2* (Isla 2005), Behavior Trees are widely used in commercial video games. In contrast to HTNs, Behavior Trees are not only responsible for decision-making but also for monitoring and the execution of behaviors. Starting with a *root* node, a Behavior Tree is built from the following nodes shown in Figure 2: *sequence, parallel, selector* (also known as *priority* or *fall-back*) that define how sub-behaviors should be executed. Additionally, the so-called *decorator* nodes can be used to extend sub-trees without modifying single behaviors. Decorators are for example *loop, counter* or *timer* nodes. Furthermore, *conditions* can be added for the selection of certain branches of a tree. Leaves of a Behavior Tree represent concrete *actions* that an agent can execute. More details on Behavior Trees can be found in Ogren (2012).

The execution of a Behavior Tree starts at the root node. In each execution cycle (so-called *tick*), the tree is evaluated and open leaf nodes are selected for execution. Due to the *parallel* node, multiple nodes can run simultaneously. In each tick, a leaf node returns its status to its parent node. The status can be *running* (as long as a behavior is executed), *success* or *failure*. Once a node succeeds or fails, its parent node reacts accordingly, either selecting another child or propagating the status further upwards. Since the conditions are constantly checked and instant switches between behaviors are possible, Behavior Trees lead to very reactive behaviors. Coming back to our previous example of entering a room, in the first tick, the Behavior Tree in Figure 2 checks the first branch of the selector node. This opens into a sequence node that checks the condition *IsReachable(door_A)*. If the condition check succeeds, the *GoToDoor* action is executed in the next tick. In case the door is closed, a failure is propagated back to the selector node which then tries to execute its second branch. If the second branch fails twice in a row (shown by the decorator node), the selector node and thus the whole tree fails.

4 Hybrid Approach

In order to combine the advantages of an HTN planner and Behavior Trees we propose replacing the lower levels of an HTN by Behavior Trees with behaviors similar to the tasks they replace. Although the purposes of HTN planners and Behavior Trees are different, from their descriptions in previous sections, it becomes obvious that there are similarities in the ways *how* a behavior is defined in each approach. This allows for an easy conversion between the domains of the two approaches. When a *method* is used to decompose a compound task in an HTN, it results in a sequence of further tasks. This corresponds to a *sequence* node in a Behavior

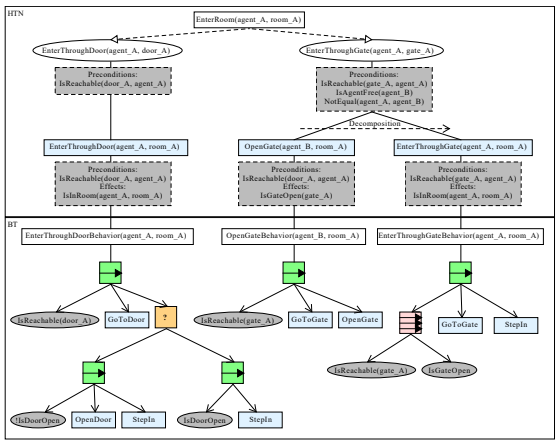


Figure 3: Hybrid representation of the *Enter a Room* example.

Tree. Being able to decompose a compound task by multiple *alternative* methods allows to replace these alternatives by a *selector* node in a Behavior Tree. *Preconditions* of methods and primitive tasks can be replaced by *condition* nodes.

The major difference between the two domains are *effects* of primitive tasks. These are required for a planner in order to be able to simulate changes in the world and plan further in advance. However, since a Behavior Tree does not plan in advance but only selects a behavior for the *current* situation, it can detect changes through direct observations of the environment and does not require predefined effects. Thus, effects of an HTN can only be represented *implicitly* in a Behavior Tree, manually ensuring that the predicted outcome of a behavior defined in a Behavior Tree corresponds to the predefined effects of an HTN task.

Returning to our example of entering a room from section 3, we can leave the high-level decision of whether an agent should enter a room through a *door* or a *gate* to the planner since this decision can be part of some high-level strategy. However, the decisions on lower levels of the entering tasks can be passed to Behavior Trees along with the actual execution of the tasks. As shown in Figure 3, the tasks *EnterThroughDoor*, *OpenGate* and *EnterThroughGate* are now changed to primitive tasks. Even though the HTN planner does not know the details of the execution of these tasks, it knows through their effects that after executing either of the selected methods *agent_A* will be in *room_A* and can continue planning with this information. On the other side, the corresponding Behavior Trees are designed in such a way that they can check certain conditions (such as *IsDoorOpen(door_A)*) at run-time and select appropriate actions that lead the agent into the room.

The planning and execution loop works as follows: the planner creates a high-level plan assigning a sequence of tasks to every agent. Each agent starts executing his first task by running the corresponding Behavior Tree. While it is running, the tree constantly checks whether the preconditions defined by the planner still hold and thus if the high-level plan is still valid. If the preconditions do not hold, the tree aborts all running behaviors and reports a failure to the planner. Additionally, it performs further checks through its nodes and makes decisions at run-time. In case the tree fails to execute the expected behavior (for example, the agent cannot find a path to the door), it also returns a failure to the planner and triggers a re-planning. Otherwise, once the tree finishes the task, it returns a success and the agent can proceed with the overall plan. If the preconditions for his next task hold, it starts running a new Behavior Tree.

5 Experiments

5.1 Experiment Setup

In order to analytically compare the execution of plans generated by the hybrid approach and a pure HTN planner (without Behavior Trees) we have created an experimental game environment using the *CryEngine*[1] game engine. We have used the engine's Behavior Trees and an adapted version of the open-source HTN planner *derPlanner*[2] (Shafranov and Champandard 2013). This planner is a *total order* planner which means that it decomposes tasks in the same order that they will be executed in later (Nau et al. 1999). In our scenario, multiple *hunters* are controlled by a centralized HTN planner. Their goal is to *lure a zombie* into a *cage*. The zombie is not controlled by our approach. Instead, it uses a simple Behavior Tree moving randomly within a *room* and attacking any hunter whenever it sees him. The room can be entered through multiple entrances, one of which is a *co-op-gate* (which requires an additional agent to hold it open). One cage entrance is inside the room so that the zombie can be lured through it into the cage. This entrance can be opened by a single agent and stays open until it is closed. The exit of the cage leads outside of the room and is also a *co-op-gate*. After luring the zombie into the cage, a hunter has to (re-) enter the room and close the cage quickly before the zombie can escape. Afterwards, the hunter can exit the room and – if required – repeat the procedure in the next room. An example of the experiment performed with the pure planner approach is shown in the video[3].

The described scenario is very dynamic due to the the actions of zombies and agents. It contains a certain degree of uncertainty because a zombie's actions are not deterministic. Also, the time required by an agent to complete a task is not known. Thus, even more uncertainty is added because, for example, one agent might not open a gate for his teammate in time. Furthermore the scenario requires a) multiple agents (at least two of them in order to exit the cage and at most three per room), b) coordination between agents, c) long-term planning (entering the room, making zombies aware and exiting it) and d) reactive behavior (reacting to zombies and proceeding through the level). It is suitable for a comparison between the hybrid approach and pure HTNs. However due to points b and c, which are not possible to achieve purely by Behavior Trees (without any coordination mech-

[1] CryEngine: www.cryengine.com

[2] derPlanner: www.github.com/alexshafranov/derplanner

[3] Experiment videos: https://bit.ly/2MW1OdY

anism on a higher level), we do not test the given scenario with pure Behavior Trees.

The domains for the given scenario are defined as 1) a pure HTN where tasks are refined up to the lowest hierarchy level as shown in Figure 1 and 2) a partial HTN describing the higher-levels of the hierarchy and multiple Behavior Trees describing lower-level tasks as shown in Figure 3.

Since a hunter can die after receiving a certain amount of damage from a zombie, he should react fast once he has the zombie's attention. Taking this into account, the hunters' goal is to lure each zombie in each room into a cage without dying. If at least one hunter dies, the execution is regarded as a *failure*. As soon as all zombies are in cages, the experiment is regarded as a *success*. In order to measure the effects of the reactivity added through Behavior Trees, we measure the percentage of execution trials in which a hunter died (failures). Additionally, we assume that the hybrid approach will be more flexible regarding small changes in the world and thus will trigger re-planning less often than the pure planner approach. For that reason, we measure the number of plan failures that trigger re-planning until the goal is reached. Here, we take into account only successful experiments. Furthermore, we assume that by not being reactive, the execution of the detailed HTN plan should take longer compared to the hybrid approach. Therefore, we measure the time required by the hunters to achieve the goal.

5.2 Results

The experiments for this work were performed with 1 to 3 rooms with each 1 zombie in it. Since the planning domain is designed in such a way that *clearing* one room requires at least 2 and at most 3 hunters, the experiments were performed with 2 to $3 * n$ hunters where n is the number of rooms. This allowed us to examine the simultaneous execution of multiple hunters (for example with 3 rooms and 9 hunters) as well the successive execution of a high amount of tasks by a few agents (for example with 3 rooms and 2 hunters). Each combination of rooms and hunters was first executed 20 times with each approach in order to measure the *failure* rate (hunters' deaths). Afterwards, the experiments were repeated until the amount of 20 *successful* executions was reached by each approach.

As described in the experiment setup, we have measured the percentage of experiment runs that failed due to a hunter's death. The results in Figure 4 show that the percentage of failed trials is, in overall, lower for the hybrid approach. We have seen that the agents using the hybrid approach were indeed more reactive. For example, instead of running towards the position that the planner *assumed* the zombie to be in (as can be seen in the videos), the hunters with Behavior Trees ran towards the zombie's *actual* position only until the zombie had visual contact with them. That way, they started moving towards the cage much earlier having more time to open the cage gate before receiving the first bit of damage. Nevertheless, they still got some damage and especially in those experiments where only a few hunters had to clear multiple rooms in a row the failure percentage was similar for both approaches. This is reflected in the values for 2 and 3 hunters in Figures 4b and 4c. However, in cases where enough hunters were available to clear multiple rooms simultaneously, the difference between the approaches was significantly higher. For example when 2 hunters had to take care of only 1 room, they almost never failed to catch the zombie (see Figure 4a and Figure 4b for 4 hunters and 2 rooms).

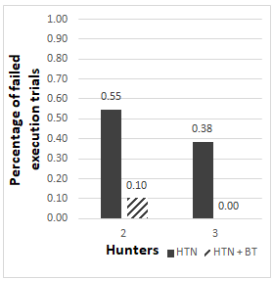
In the next step, we have measured the number of plan failures that triggered a re-planning. Assuming that most plan failures occur due to small changes in the world, we supposed that plans should fail less often and only due to severe changes when using Behavior Trees for the detection of such low-level changes and reacting to them. Also, agents with Behavior Trees should be more flexible and reach the goal faster. Figure 6 shows that in almost all cases the time needed by the hunters to reach the goal was shorter for those using the hybrid approach. Similarly, as Figure 5 shows, the number of plan failures was always higher for the agents that used a planner only. During the experiments we could see that, for example, agents with the hybrid approach were able to change their behavior slightly by stepping aside or opening a closed door, whereas agents without Behavior Trees would trigger a new plan. The difference became more obvious as the number of agents grew, as we can see in Figure 5c. This is because with a higher number of agents more coordination on the higher hierarchy levels was required from the planner where at the same time more disturbances on the lower levels appeared. For example, the agents were more likely to cross or block each other's path and manipulate exit objects. At this point, it is noteworthy that the experiments were meant to show the differences *between the two approaches* for different setups. Thus, the differences between the results of different number of hunters for *one* approach are not relevant since they strongly depend on the domain.

6 Conclusions and Future Work

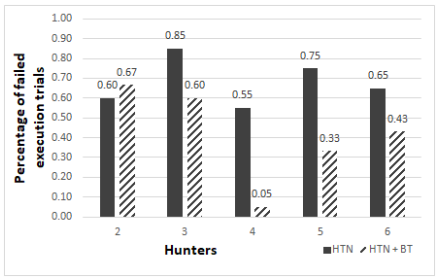
This work proposes a hybrid approach to planning and execution which allows for building long-term strategies while staying reactive to changes in the environment. This is achieved through a combination of a Hierarchical Task Network planner for high-level planning and Behavior Trees for low-level decision making and execution. The centralized planner is able to create plans for multiple agents without going into details of *how* the low-level tasks will be executed. Every low-level task is encoded as a separate Behavior Tree which decides how to execute the task while monitoring the preconditions given by the planner as well as further changes in the environment at run-time.

The hybrid approach is tested in a video-game environment and compared against a pure HTN planner. Although both approaches share the same high-level planning domain, there are visible advantages when using the hybrid approach. The behaviors executed with the hybrid approach are more flexible and thus fail less often. Furthermore, agents using the hybrid approach can reach their goals faster avoiding some unnecessary actions.

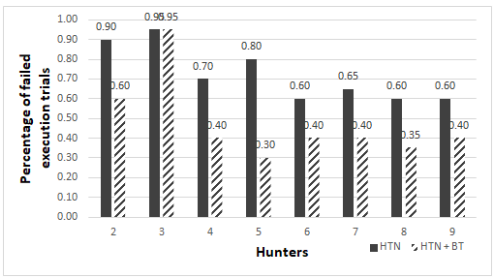
The experiments performed in this work show that there are many ways to further extend this work. One possibility is to translate HTNs into Behavior Trees automatically. Alternatively, Behavior Trees that replace the lower levels of an



(a) 1 room with 1 zombie.

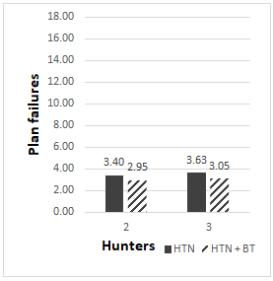


(b) 2 rooms with 1 zombie each.

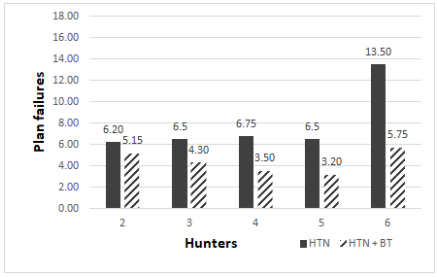


(c) 3 rooms with 1 zombie each.

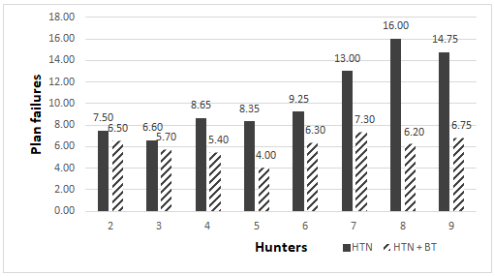
Figure 4: The percentage of failed execution trials which occurred due to a hunter's death that occurred trying to catch 1,2 and 3 zombies by varying numbers of hunters using a pure HTN planner and the combination of an HTN planner with Behavior Trees.



(a) 1 room with 1 zombie.

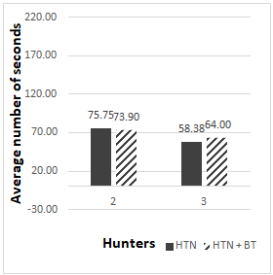


(b) 2 rooms with 1 zombie each.

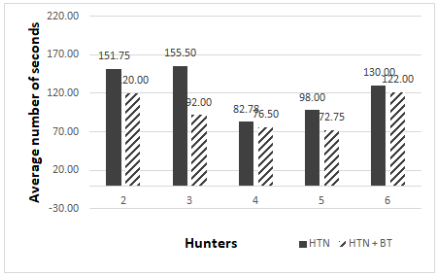


(c) 3 rooms with 1 zombie each.

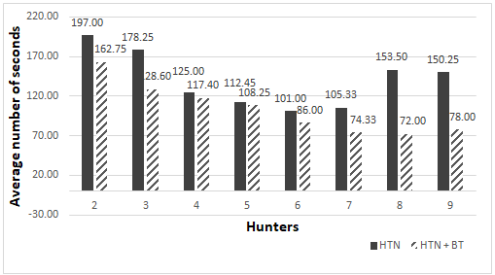
Figure 5: The average number of plan failures that triggered re-planning before the goal was reached. Measured with 1,2 and 3 zombies and a varying number of hunters using a pure HTN planner and the combination of an HTN planner with Behavior Trees.



(a) 1 room with 1 zombie.



(b) 2 rooms with 1 zombie each.



(c) 3 rooms with 1 zombie each.

Figure 6: The average time in seconds required to reach the goal. Measured with 1,2 and 3 zombies and a varying number of hunters using a pure HTN planner and the combination of an HTN planner with Behavior Trees.

HTN could be learned from scratch (similar to (Colledanchise, Parasuraman, and Ögren 2015)). This could be done, for example, taking the preconditions and effects of higher-level HTN tasks into account. Another important aspect to look into is the extensibility of this approach to bigger search spaces and higher numbers of agents, for example in an RTS game. The setup used here was quite simple, so that the depth of the HTN was shortened by the Behavior Trees by approximately 3 levels (from a maximum depth of 7). In more complex environments, the HTNs might be much deeper. It is worth investigating the differences in the computational time as well as behavioral changes when replacing different levels of an HTN with Behavior Trees.

References

Bolander, T. 2017. A gentle introduction to epistemic planning: The del approach. *Theoretical Computer Science* 243:1–22.

Colledanchise, M.; Almeida, D.; and Ögren, P. 2016. Towards blended reactive planning and acting using behavior trees. *arXiv preprint arXiv:1611.00230*.

Colledanchise, M.; Parasuraman, R.; and Ögren, P. 2015.

Learning of behavior trees for autonomous agents. *arXiv preprint arXiv:1504.05811*.

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D. E.; and Washington, R. 2003. Incremental contingency planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling workshop on Planning under Uncertainty and Incomplete Information*, 38–47.

Dini, D. M.; Van Lent, M.; Carpenter, P.; and Iyer, K. 2006. Building robust planning and execution systems for virtual worlds. In *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference*, 29–35.

Engesser, T.; Bolander, T.; Mattmüller, R.; and Nebel, B. 2015. Cooperative epistemic multi-agent planning with implicit coordination. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling 3rd Workshop on Distributed and Multi-Agent Planning*, 68–76.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. volume 2, 189–208. Elsevier.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2015. Backward-forward search for manipulation planning. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 6366–6373. IEEE.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

Hilburn, D. 2014. Simulating behavior trees: A behavior tree/planner hybrid approach. In *Game AI Pro: Collected Wisdom of Game AI Professionals*, 99–112. CRC Press.

Isla, D. 2005. Managing complexity in the halo 2 AI system. In *Proceedings of the Game Developers Conference*.

Jacopin, E. 2014. Game AI planning analytics: The case of three first-person shooters. In *Proceedings of the 10th Artificial Intelligence and Interactive Digital Entertainment Conference*, 119–124.

Menif, A.; Jacopin, É.; and Cazenave, T. 2014. SHPE: HTN planning for video games. In *Computer Games*. Springer. 119–132.

Muise, C. J.; Belle, V.; Felli, P.; McIlraith, S. A.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Planning over multi-agent epistemic states: A classical planning approach. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 3327–3334.

Nau, D., and Champandard, A. 2012. Inside hierarchical task network planners. http://aigamedev.com/premium/interview/htn-planners/.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence*, volume 2, 968–973. Morgan Kaufmann Publishers Inc.

Neufeld, X.; Mostaghim, S.; Sancho-Pradel, D.; and Brand, S. 2017. Building a planner: A survey of planning systems used in commercial video games. *IEEE Transactions on Games*.

Ogren, P. 2012. Increasing modularity of UAV control systems using computer game behavior trees. In *AIAA Guidance, Navigation, and Control Conference*, 4458.

Orkin, J. 2006. Three states and a plan: the AI of FEAR. In *Proceedings of the Game Developers Conference*.

Palma, R.; González-Calero, P. A.; Gómez-Martín, M. A.; and Gómez-Martín, P. P. 2011. Extending case-based planning with behavior trees. In *Proceedings of the 24th International Florida Artificial Intelligence Research Society Conference*, 407 – 412.

Rabin, S. 2009. #define GAME_AI. In *Proceedings of the Game Developers Conference*.

Shafranov, A., and Champandard, A. 2013. Planning domains and compiling htn to c++. http://aigamedev.com/premium/interview/plan-compilation/.

Soemers, D. J., and Winands, M. H. 2016. Hierarchical task network plan reuse for video games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 1–8. IEEE.

Vassos, S., and Papakonstantinou, M. The simpleFPS planning domain: A PDDL benchmark for proactive NPCs. In *Proceedings of the 7th Artificial Intelligence and Interactive Digital Entertainment Conference Workshop on Intelligent Narrative Technologies*, 92 – 97.

Weber, B. G.; Mawhorter, P.; Mateas, M.; and Jhala, A. 2010. Reactive planning idioms for multi-scale game AI. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 115–122. IEEE.

Weber, B. G.; Mateas, M.; and Jhala, A. 2011. Building human-level AI for real-time strategy games. In *Proceedings of the AAAI Fall Symposium: Advances in Cognitive Systems*, volume 11, 329 – 336.