Procedural Creation of Behavior Trees for NPCs

Robert Fronek, Barbara Göbl, and Helmut Hlavacs(✉)

Entertainment Computing, University of Vienna, Vienna, Austria
helmut.hlavacs@univie.ac.at

Abstract. Based on an emerging need for automated AI generation, we present a machine learning approach to generate behavior trees controlling NPCs in a "Capture the Flag" game. After discussing the game's mechanics and rule set, we present the implemented logic and how trees are generated. Subsequently, teams of agents controlled by generated trees are matched up against each other, allowing underlying trees to be refined by learning from victorious opponents. Following three program executions, featuring 1600, 8000 and 16000 matches, highest scoring trees are presented and discussed in this paper.

Keywords: Behavior trees · Machine learning · Procedural creation

1 Introduction

Despite its major impact on gaming experience, creating artificial intelligence (AI) for games is increasingly pushed towards late stages of the development process, freeing resources for easily marketable features such as graphics [4]. Thus, one can currently observe a trend towards automatically generated AI in the industry [6]. These approaches allow for procedural generation of opponents that are still able to display different behaviors.

The following paper presents a program that procedurally generates behaviour trees (BT) for a simple "Capture the Flag" game. Games have a fixed set of rules and possible interactions, making them ideal candidates to test the capabilities of an AI [8]. The quality of an AI is evaluated by matches between randomly generated BTs. When a BT is defeated, it is discarded and replaced by a new one, which, after a sufficient number of matches, eventually leads to a BT that is superior to a large number of other, randomly generated BTs. The game, despite its turn-based approach, shares various aspects with real-time strategy (RTS) games, such as commanding units, strategic decision making and aiming for both short- and longterm goals [7].

2 Related Works

Several previous works discuss applications of automated BTs. Lim et al. [3] managed to defeat the AI of the game "DEFCON" through evolution and combination of BTs. Abiyev used BTs to control soccer robots [1]. Dromey points out

Published by Springer Nature Switzerland AG 2020
N. J. Nunes et al. (Eds.): ICEC 2020, LNCS 12523, pp. 285–296, 2020.
https://doi.org/10.1007/978-3-030-65736-9_26

that BTs are a good alternative to automated AI, due to a clear connection of requirements and formal specifications [2]. This allows to generate requirement-based BTs.

3 Game Mechanics

The game is played on a board comprising a total of 56 (7×8) fields. Each field can only be occupied by one agent. If another agent, either ally or enemy, is already placed on a field an agent cannot move onto it. There are three types of terrain for each field:

- **Open:** Every agent can move onto this field, there are is no bonus or penalty.
- **Blocked:** No agent can move onto this field. Agents may shoot across the field with no bonus or penalty.
- **Cover:** Every agent can move onto this field. Additionally, damage is reduced when placed on this field if attacked from the front. Damage from melee attacks is not reduced, neither is damage from attacks originating from behind or either side.

Figure 1 shows the game's board. Two fields, symmetrically placed, are blocked (marked "x"). Additionally, there are 4 fields of type "cover" (marked: "c") in the field's center as well as next to the blocked fields. The flag's position is marked "F". Numbers indicate x- and y-coordinates as used by the game's internal logic.

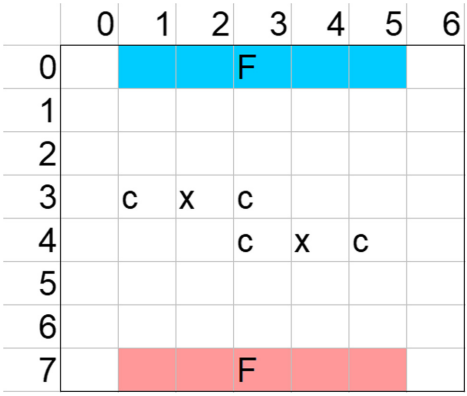

Fig. 1. Empty board (Color figure online)

Each team starts off with 5 units (agents) positioned on the top (colored in blue) and bottom row (colored in red) of the board. A team wins if one of its agent reaches the opposing team's flag. If no agent has reached the opposing team's flag within 100 turns, the game results in a draw.

Sweetser et al. [9], define (software) agents as goal-driven entities that are aware of their environment. In this work, agents operate based on a number of parameters as described below. The AI's success determines whether chosen actions are suitable to support the game's goal.

Agents initially possess 25 health points (HP) and can perform ranged and melee attacks. If the HP value decreases to 0 or less it is defeated and removed from the game. Ranged attacks can be carried out 2 times before ammunition needs to be reloaded. Reload is either initiated by explicitly performing the corresponding action or happens automatically if a ranged attack is conducted without ammunition, thus replacing the attack action. There are 9 possible actions for an agent. Each turn, an active agent (i.e. an agent that has not been defeated yet) performs one of the following actions:

- **Move towards an enemy:** The agent looks for nearby enemy agents on the field. If none are left, he moves towards the opposing team's flag instead. Distances are measured in fields and the agent may move one field horizontally or vertically per turn. Vertical movements are prioritized over horizontal movements until the opposing agent is positioned at the same row. If a field is blocked, the agent moves sidways at a 90° angle to the initially calculated direction. If none of these movements is possible, the agent can perform no more action in this turn and stays in place.
- **Move towards the enemy's flag:** The agent moves one field towards the opposing team's flag. Similar to the previous action, vertical movement is prioritized if possible and necessary.
- **Move towards the own team's flag:** The agent moves one field towards their own team's flag. Again, vertical movements precede horizontal movements.
- **Heal**: The agent stays in place and restores a small amount of HP.
- **Shoot enemy agent:** The agent shoots the nearest enemy agent. Damage is higher, the closer the other agent is. Shot agents take increased damage if attacked from their sides (double damage) or behind (triple damage). In both cases, "Cover" fields do not reduce damage. If an agent is in cover, frontal damage is reduced by half. If no ammunition is left, the agent reloads instead.
- **Perform melee attack:** The agent attacks an enemy agent on an adjacent field, doing damage dependant on the attack's direction. Sideway attacks do double damage while attacks from behind immediately defeat the attacked agent. If there is no enemy on adjacent fields, the agent moves towards an enemy instead.
- **Flank:** The agent preferably tries to move on a field directly behind an opposing agent, or, alternatively, to the side of an opposing agent.
- **Reload:** Ammunition is restored to 2.
- **Idle:** The agent stays idle.

4 Logic

Each team generates a BT to decide which actions an agent performs. The BT is a binary tree where all internal nodes are logic nodes and leaf nodes

represent actions. Each logic node represents a comparison with a randomly chosen operator and a threshold. One of the following comparisons may be assigned:

- Distance from the own flag to its closest enemy agent
- Distance from the closest enemy agent to self
- HP of self
- Distance from self to the enemy's flag
- Remaining ammunition
- Whether self is in cover
- Whether closest enemy agent is in cover
- Number of nearby allied agents (teammates)
- Number of nearby enemy agents

Possible operators are $\leq$(less or equal) and $\geq$(greater or equal). Thresholds are randomly assigned from predefined ranges. Distances are assigned between a range from 1–15 as 15 is the maximum distance on a 7×8 board (based on the 1 norm). Agents' HP range from 0–25, ammunition ranges from 0–2. Checking whether the agent itself or enemy agent's are positioned on "Cover" needs no threshold as it is a logical query. Nevertheless, 0.5 is assigned to avoid NULL values. To query for nearby enemy agents, two parameters are needed. A threshold, assigned between 1 and 4, limits the amount of agents that are looked for. The second parameter defines the maximum distance of fields for the search, possible values range from 1–6.

Trees are generated as follows: 4–12 actions are randomly assigned to leaf nodes, with corresponding 3–11 internal logic nodes holding randomly assigned comparisons and operators. Subsequently, a threshold and, if needed, a second parameter are assigned randomly. Tree traversal starts by evaluating the logical statement in the root node. If true, the left child is visited, otherwise the right child is visited. If the next node is a logic node, traversal continues as described, if it is a leaf node, the corresponding action is performed. Active agents decide which action to perform by traversing the tree each turn.

5 Methodology

Ponsen et al. [5] show, that dynamic AIs which adapt based on victorious strategies are superior to static AIs. Thus, defeated AIs should learn from victorious BTs and try to adapt to their strategies.

At first, BTs are generated for 32 teams. Teams are then paired against each other and initialized with a score of 1. Subsequently, each team keeps count of their victories. Victorious teams increment their score by 1 plus half of the points (rounded down) of the defeated team, except for victors in the first round who double their score to 2.

5.1 Generation of a New Tree (No Draw)

The defeated team's tree is generated based on both opposing parties' trees. Size of the tree is randomly assigned from 3 options: size of the victorious team's tree,

size of the defeated team's tree or a randomly assigned new size. Similarly, each node is randomly inherited from the corresponding node from either tree or randomly generated. If the algorithm tries to base a node on a pre-existing tree with smaller size holding no corresponding node, it falls back to the other tree's nodes, or, ultimately, random generation of a new node.

Generation of new trees is based on weighted decisions. First, the upper limit l of a range is determined by the victorious team's score (S_v), the defeated team's score (S_d) and a constant of 5, to allow for randomly assigned nodes, as follows: $l = S_v \times 3 + S_d + 5$. A random number x is generated within the range of 0–l. If $x < S_v \times 3$, nodes are inherited from the victor's tree, if $S_v \times 3 \leq x < (S_v \times 3 + S_d)$ nodes are inherited from the defeated tree. If the random value is greater or equal to the latter limit ($x \geq (S_v \times 3 + S_d)$), a randomly generated new node is assigned. This enables defeated teams to "learn" from victors but still allows for improvements based on random adaptations.

Example: A team with a score of 7 ($S_v = 7$) defeats a team with a score of 2 ($S_d = 2$). This results in a range of 0–27 for the randomly generated number x. If $0 \leq x < 21$, the node is inherited from the victorious team's tree. If $21 \leq x < 23$, the node is inherited from the defeated team's tree. Finally, if $x \geq 23$ a new node is generated randomly. These newly generated trees are assigned a score S, where $S = 1 + \lfloor (S_v + S_d)/10 \rfloor$.

5.2 Generation of a New Tree (Draw)

If no team is victorious after 100 turns, the game results in a draw. Typically, if a team wins eventually, a game lasts about 15–40 turns. Thus, probability of a draw is higher if a game goes on beyond 40 turns. Considering the random generation of trees, draws might e.g. be caused by trees that never move an agent towards the enemy flag. In this case, both trees are replaced, where one tree is generated based on both team's trees as described above (one tree is assumed as victor). The other tree will be generated from scratch and initialized with a score of 1.

5.3 In-between Matches

After each match, the victorious team's score is checked against the current high score. If the team has a higher score, the new high score is saved and it's underlying tree is saved for output after completion of all matches. Additionally, the previously highest scoring tree is now ranked second and will also be outputted. Eventually, the vector holding all teams will be shuffled to create new pairings. After a predefined number of matches, the program terminates and outputs the two highest scoring trees and general statistics (see Table 1).

6 Results

The program was executed 15 times, 5 times each for 100 rounds (1600 matches), 500 rounds (8000 matches) and 1000 rounds (16000 matches). The most

superiour trees of each set are discussed below. Table 1 provides further data on program execution and results, such as number of victories, draws and high scores (HS).

Table 1. General statistics

Rounds	100			500			1000		
Run	Victories	Draws	HS	Victories	Draws	HS	Victories	Draws	HS
1	1451	149	54	5913	2087	34	13238	2762	50
2	1425	175	43	6887	1113	49	14027	1973	75
3	1427	173	41	7577	423	61	13308	2692	60
4	1362	238	47	6683	1317	69	13004	2996	56
5	1458	142	50	6776	1224	48	14026	1794	53
Avg	1224.6	175.4	47	6767.2	1232.8	52.2	13520.6	2479.4	58.8
%	89,0%	11,0%		84,6%	15,4%		84,5%	15,5%	

7 Analysis of Trees

Table 1 provides some first insights. Despite a higher number of matches, results show a roughly equal rate of draws. In a match between two "competent" AIs, a draw is rather unlikey, due to the first-move advantage. This suggests, that a rather constant rate of "inapt" AIs (AIs that do not move towards the opponent's flag) is generated during program execution.

Program execution characterized by higher scores result in less matches ending in a draw. It can be safely assumed, that this is due to early generation of a superiour AI that subsequently defeats its opponents and serves as a basis for opponent's to learn from. The first run featuring 500 rounds illustrates this issue very well, as a low high score is accompanied by a high number of draws, which suggests that random generation of BTs did not produce a superiour AI for some time.

The gathered data during program execution also shows that a higher amount of rounds does not result in higher scores. One could naively assume that 5 times as many rounds result in at least 5 times higher scores - or even more, as a superious AI might gather more points from defeated, but previously successful, opponents. A possible explanation might be that defeated AIs learn rather quickly from superior AIs which eventually leads to equally "competent" opponents.

7.1 Analysis of Superior Trees (100 Rounds)

Tree 1 (Fig. 2) features a "Enemy in Cover" query in its root node (true represented by 1, false by 0). However, the query is more likely to return false, which

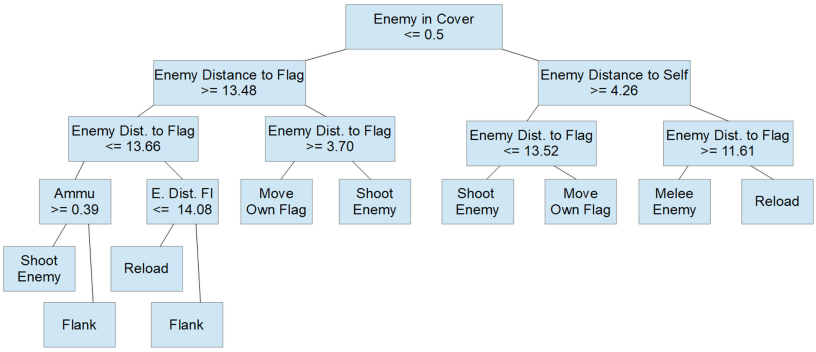

Fig. 2. Tree 1–100 rounds, run 1, score: 54

results in more frequent traversal of the left side of the tree. Please keep in mind, that the tree is traversed according to the following rules: if a query returns true, the left child is visited, otherwise the right child.

"Enemy Distance to Flag" is always less than 13 as long as there are active, opposing agents, leading to traversal of the right children of this node. This results in a mostly defensive behaviour: agents wait, in close distance to their own flag, for opposing agents to advance and shoot them when in reach. As soon as all opposing agents are defeated, querying "Enemy Distance to Flag" returns a value of 1000, leading the agents to try to flank. As there are no further opponents and the flag can not be flanked, agents move directly towards the other team's flag instead. Similar behaviour can be observed when opponents are in cover: agents move towards their own flag or reload ammunation. As there are is no cover in immediate distance to the flag, the AI may easily defend its flag.

Tree 2 (Fig. 3) mostly leads to melee attacks: the root node likely returns false as all allied agents still have to be active and nearby. Agents move towards the enemy if no enemy is nearby and subsequently move towards the flag if no opposing agent is still active - resulting in a simple, but efficient tree.

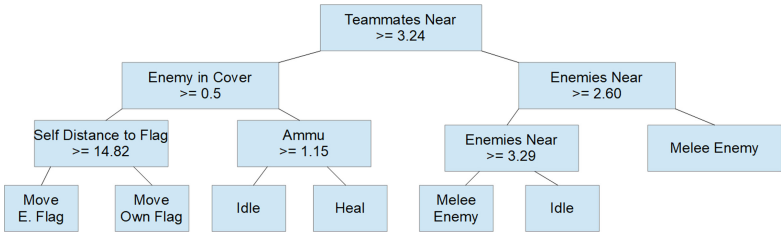

Fig. 3. Tree 2–100 rounds, run 2, score: 43

The right side of **Tree 3** (Fig. 4) is rarely traversed, as an agent's HP are likely higher than 3.76. The AI lacks opportunity to use ammunition and will therefore mostly perform melee attacks unless many enemies (>3.28) are nearby and the agent heals itself.

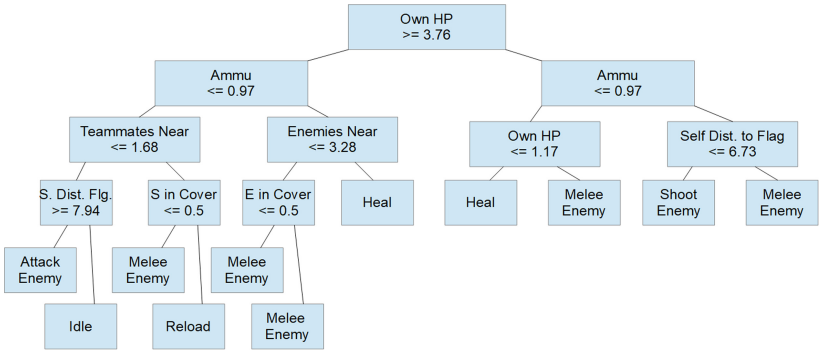

Fig. 4. Tree 3–100 rounds, run 3, score: 41

Tree 4 is good example of shirking responsibility: as long as there are nearby allied agents, agents stay idle. This leads to agents on the outer borders of the formation to attack enemys first, while remaining agents, initially positioned in the middle, attack later on (Fig. 5).

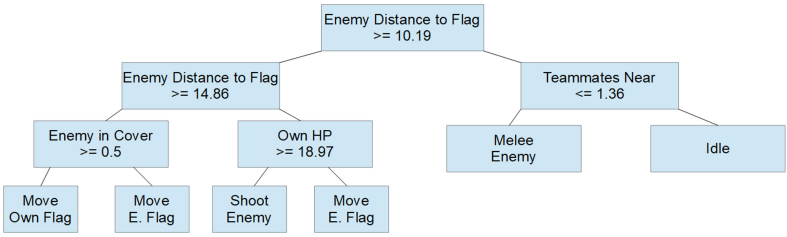

Fig. 5. Tree 4–100 rounds, run 4, score: 47

7.2 Analysis of Superior Trees (500 Turns)

Similiar to some previous trees, **tree 6** (Fig. 6) will mostly perform melee attacks. If many allied agnets are nearby, they might stay in cover. Nevertheless, as fields providing cover are limited, some agents will continue to attack.

In contrast, **tree 7** (Fig. 7) displays a new approach: the team will split up and try to reach the enemy's flag as fast as possible.

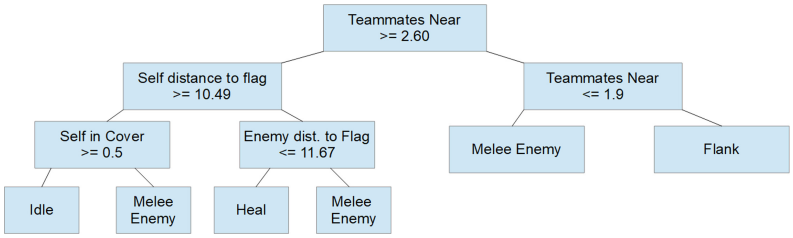

Fig. 6. Tree 6–500 rounds, run 1, score: 34

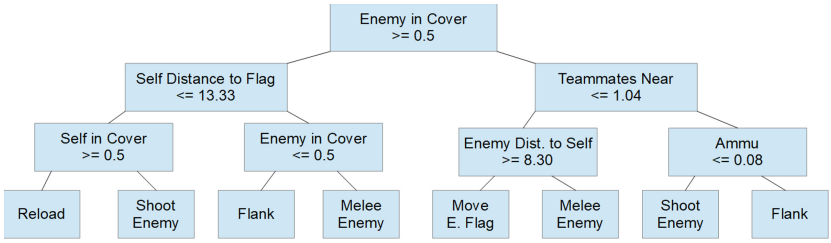

Fig. 7. Tree 7–500 rounds, run 2, score: 49

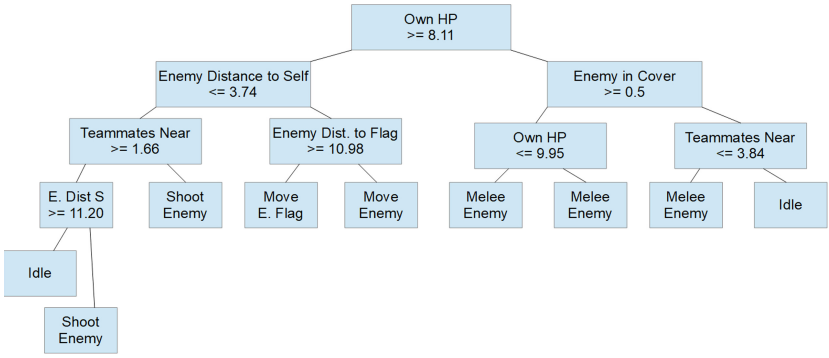

Fig. 8. Tree 8–500 rounds, run 3, score: 61

Tree 8 (Fig. 8) will likely resort to melee attacks as well. However, if HP are low, ranged attacks are more likely.

Tree 9 (Fig. 9) displays a more balanced behavior. Agents peform both melee and ranged attacks, except in case of many nearby enemies, which lead to a retreat towards the own flag. This trees displays strategic, team-based behavior and scores highest during execution with 500 rounds and its high score is only topped by one tree in the 1000 round scenario.

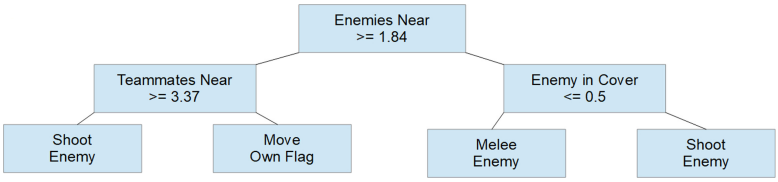

Fig. 9. Tree 9–500 rounds, run 4, score: 69

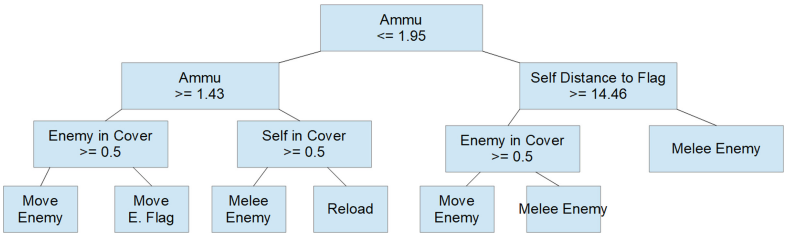

Fig. 10. Tree 11–1000 rounds, run 1, score: 50

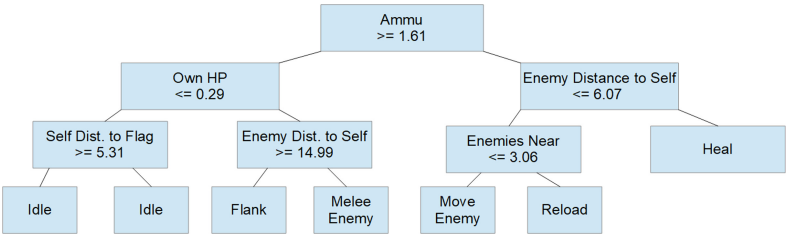

Fig. 11. Tree 12–1000 rounds, run 2, score: 75

7.3 Analysis of Superior Trees (1000 Rounds)

Tree 11 (Fig. 10) and **tree 12** (Fig. 11) display well known patterns resulting in melee attacks. Amount of ammunition never lowers, as agents never shoot and the distance to the flag will always be less than 14.

Tree 13 (Fig. 12) also mostly results in melee attacks. If many enemies are nearby, agents will constantly reload ammunition until other allied agents have dealt with aforementioned enemies.

On first inspection, **tree 14** (Fig. 13) might favor "move" and "reload" actions. However, due to the randomized definition of "near", the leaf node instructing agents to peform a melee attack instruction can be reached and lead to a victory for the team.

Tree 15 (Fig. 14) displays a more differentiated behavior. As long as enemies are not nearby, agents will move towards them based on the "Melee Enemy" instruction. Otherwise, agents attack enemies depending on whether they are

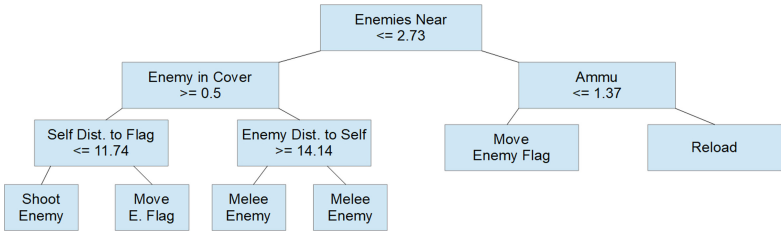

Fig. 12. Tree 13–1000 rounds, run 3, score: 60

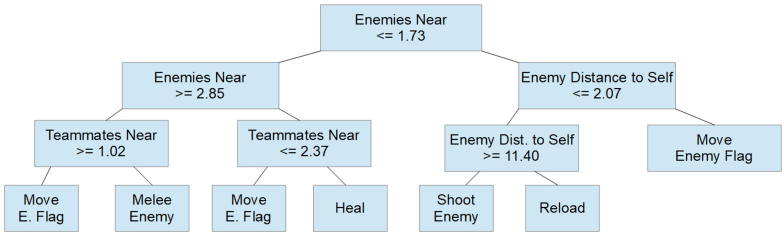

Fig. 13. Baum 14–1000 rounds, run 4, score: 56

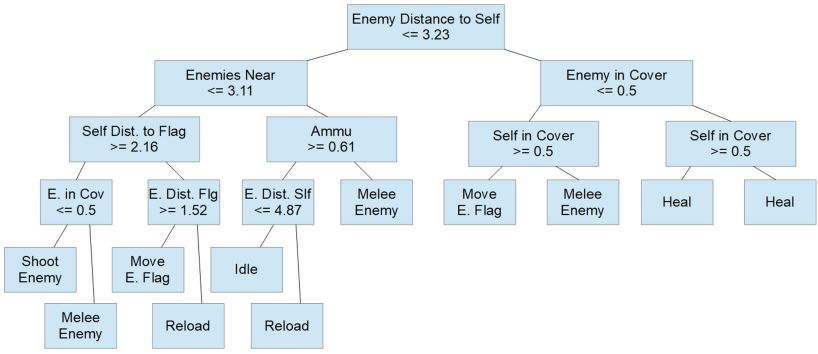

Fig. 14. Tree 15–1000 rounds, run 5, score: 53

in cover. If enemies are not in cover, agents shoot them. In case enemies are in cover agents resort to melee attacks, which do more damage to enemies in cover.

8 Conclusion

There is a clear trends towards "simple" trees, which are somewhat obfuscated by hardly traversed branches. Further work might take a closer look on these rarely visited branches and nodes and deliver further insights. Furthermore, trees could be refined by reordering nodes, as suggested by Utgoff et al. [10]. This helps to address the issue of branches that could not be reached previously.

If actions can not be performed, resorting to more suitable, alternative actions leads to lesser draws. However, this also favors rather simple trees performing the same action repeatedly. Randomly generated trees also often feature branches that cannot be reached, leading to frequent traversal of branches featuring effortless and simple actions in their leaf nodes. These trees are also more likely to be generated during early stages and thus, pass on this behavior to other trees learning from them.

References

1. Abiyev, R.H., Akkaya, N., Aytac, E.: Control of soccer robots using behaviour trees In: 2013 9th Asian Control Conference (ASCC), pp. 1–6. IEEE (2013)
2. Dromey, R.G.: Using behavior trees to model the autonomous shuttle system. In: 3rd International Workshop on Scenarios and State Machines: Models, Algorithms and Tools, (SCESM04), Edinburgh. IET (2004)
3. Lim, C.-U., Baumgarten, R., Colton, S.: Evolving behaviour trees for the commercial game DEFCON. In: Di Chio, C., et al. (eds.) EvoApplications 2010. LNCS, vol. 6024, pp. 100–110. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12239-2_11
4. Nareyek, A.: AI in computer games. Queue **1**(10), 58–65 (2004)
5. Ponsen, M., et al.: Automatically generating game tactics through evolutionary learning. AI Mag. **27**(3), 75–75 (2006)
6. Riedl, M.O., Zook, A.: AI for game production. In: 2013 IEEE Conference on Computational Inteligence in Games (CIG), pp. 1–8. IEEE (2013)
7. Robertson, G., Watson, I.: A review of real-time strategy game AI. AI Mag. **35**(4), 75–104 (2014)
8. Schaeffer, J.: A gamut of games. AI Mag. **22**(3), 29–29 (2001)
9. Sweetser, P., Wiles, J.: Current AI in games: a review. Aust. J. Intell. Inf. Process. Syst. **8**(1), 24–42 (2002)
10. Utgoff, P.E., Berkman, N.C., Clouse, J.A.: Decision tree induction based on efficient tree restructuring. Mach. Learn. **29**(1), 5–44 (1997)