Creating and Executing Abstract Strategy for Arma 3 AI with a Hierarchical Task Network

Garrett Finn
finnx149@umn.edu

December 18, 2019

Abstract

Hierarchical Task Networks are ideal solutions for Real-Time Strategy games. Despite being a military-simulation sandbox rather than an RTS, the same principles for strategic thinking and execution can be applied to Arma 3 with a hierarchical task network. This solution will provide greater challenge to Arma 3 players with minimal, and in some cases positive, performance impact. Literature discussing various HTN implementations for many different games or situations are reviewed and acknowledged as inspiration. Results from performance tests on the author's Arma 3 HTN are presented and analyzed, showing minimal performance cost and presenting a use for the system to be used to increase average game performance. Finally, future work is proposed to expand the usefulness and realism of the HTN by incorporating more modern military doctrines.

1 Introduction

Arma 3 is the premier military-simulation sandbox game on the PC market. It offers a wide range of options for simulating large scale combined arms battles. The primary feature is cooperative missions created by community members. A vast number of gaming clans have been formed that run weekly cooperative missions with attendance numbers that can reach the hundreds. However, each of these missions need a disproportionate amount of time to create. A two hour mission make take five to ten times the development time. A large portion of this development time is devoted to writing AI plans and making it dynamic and reactive to player actions.

Arma 3 also features a free-form AI unit control system that can react to the world around it. However, these AI units must be supplied a hand crafted series of waypoints or commands in order to accomplish complicated tasks. The AI units are tactically minded, but cannot think or reason strategically. There exists a need for a system that can algorithmically and intelligently create and execute strategic plans for AI units to provide more realistic opponents for player-controlled units in cooperative missions. We propose a solution to this

need through a Hierarchical Task Network (HTN) that can take high-level abstract tasks, such as "Attack this hill" and deconstruct it via its hierarchical networks into a plan of atomic actions. A force of AI units controlled by our HTN can plan an assault based on a list of given objectives, delegating tasks to appropriate groups and units. This solution is intended to reduce the amount of time needed to develop cooperative missions for the Arma 3 community, as well as make the frequently incompetent base-game AI a more dangerous opponent.

Additionally, the nature of Arma 3's simulation engine as well as its age makes its average performance severely lag behind its contemporaries. A "good" average frames per second (FPS) for Arma 3 is considered to be anything higher than 30 FPS (for other PC games, the "good" is 60 FPS). A possible side effect of our "smarter" AI system is to reduce the number of AI units required to provide a significant challenge to players in a mission, and thereby increase the overall game performance and experience for players.

2 Related Work

The problem of real-time strategy (RTS) game playing AI is not considered a solved one. Unlike chess, top human players are still able to best AI players consistently. RTS game environments demand a balance between deliberate and reactive actions, a demand that has not been totally met with artificial intelligence. For our implementation of a simple RTS-style algorithm for controlling units in the military simulation Arma 3, we look for inspiration and information from papers presenting a variety of approaches addressing real-time adaptive planning in dynamic environments.

In his thesis, Synnaeve [7] implements a StarCraft-playing bot that anticipates the enemy's strategy through its analysis of the StarCraft meta and principles of Bayesian uncertainty. Like many game AI, how Synnaeve's bot acts is determined by finite state machines (micro), while an overhead Bayesian "arbitrator" analyzes information available to predict the enemies future force composition, and therefore its strategy (macro). Players execute strategies in StarCraft by building complimentary force compositions, so predicting the opponent's force composition leads to a finite set of strategies available to them. The effectiveness of this approach is shown in Synnaeve's 12th place rank in a worldwide StarCraft bot ladder.

Smith and Nau discuss the difficulties of game-tree searches for imperfect-information games [6]. They observe the card game Bridge, not through what actions can be taken for any given hand, but for what plans can be created from that hand. A key aspect is that Smith and Nau propose a model that does not necessarily plan out the enitre game, only sections. They use a "decision-tree" rather than a game-tree. Nodes are either "decision nodes" where the agent must make a decision, or "external-agent nodes", where other players make decisions. Leaf nodes are either game end states or states where no decision can be made. Granted, this will not work for a real-time environment, but the ideas presented can be adapted for other imperfect-information games.

Hoang et al. present a Hierachical Task Network (HTN) solution, an implementation of Goal-Oriented Action Planning (GOAP) [4]. GOAP is based on the Stanford Research

Institute Problem Solver (STRIPS), an alternative to the popular state machine that is able to more dynamically find solutions for given problems. An HTN works by taking a knowledgebase of a virtual world (i.e. terrain, objects, obstacles, and objectives) in order to solve a problem, using the highest level tasks to reach conclusions. The HTN algorithm will break each task into their simplist forms (actions). The hierarchy is between two levels of tasks: "compound tasks" (tasks made of more than one action) and "primitive tasks" (tasks that consist of only one action and cannot be further broken down). Each component of an HTN, called a method, has 3 sections: the resulting state, the preconditions, and the actions needed to accomplish the resulting state.

Hoang et. al. implemented an HTN system for bots in the first-person shooter Unreal Tournament. The HTN coordinates and plans strategy for the team of bots, while micro actions are controlled by finite state machines. This is of particular interest to our project to implement a similar system in Arma 3, where micro actions (tactics) are already controlled by the game.

Gorniak and Davis approach attempts a broader application of the Hierarchical Task Network (HTN), striving to create an algorithm capable of handling more generic games [3]. They implement an HTN. Gorniak and Davis describe an extension for HTN that implements synchronization between agents. Their goal here is demonstrated on a virtual police squad: the policemen work in pairs to subdue criminals and navigate hostile environments. The highlight of their algorithm is their domain compilation and time slicing of the planning to increase performance. Since Gorniak and Davis intend for their algorithm to be generic, they cannot have a static set of states or problems, those will be different for every game. But there will be a static domain that can be determined (and optimized) during compilation. Additionally, with time slicing, the algorithm can pause and resume at will. This means that it can generate a new instruction each frame instead of generating an entire plan each frame, greatly reducing the performance impacts on a game's real time environment.

Baxter also uses a hiearchical planning model to achieve strategic actions, particularly for military-style command structures [1]. "Commander" agents plan and execute broad strategic plans, while more local agents plan and execute narrow tactical plans. A Commander agent does not waste processing power planning for a newly discovered enemy unit because the local agent plans for and executes to solve that problem itself. Higher level planning remains focused on high level strategy, while low level planning only concerns itself with its individual goals. Local agents can also re-plan when they encounter an interruption (i.e. encountering a new enemy, a route ordered to them being too dangerous, a route made safer by lack or destruction of enemies). This system is an ideal goal for what a perfect implementation for Arma 3 could be.

Bjarnolf et al. describe an approach to threat analysis using GOAP [2] focusing on predicting the consequences of actions taken by a planning agent. In a complex and dynamic environment like the real world, an agent can easily do actions based on a list of preconditions and know what some of the consequences will be, but it cannot know what unintended consequences it might have, as not everything can be planned for in compilation. Bjarnolf et. al. discuss the idea of an agent's situational awareness, and how it can be increased by

sharing information between agents in a hierarchical plan. They propose an extension of the GOAP model to better incorporate "Commander's Intent" or the desired effect of an action that might have several outcomes (i.e. the difference between a squad of soldiers moving to a position with the intent to provide fire support for another squad versus a squad moving to a position so as to assault it. Both would require different localized planning to achieve successfully).

Ontanon et al. outline the ineffectiveness of search-based AI techniques to handle the huge search spaces of RTS games [5]. They present a case-based planning architecture, constructing cases from system observation of expert human players. By observing expert players, it builds a library of behavior it searches when presented with problems. This library is a much smaller space to search than the game-tree, providing a much more efficient method of playing the game. It is not dependant on pre-scripted plans made by game developers, and therefore not susceptible to the same exploitations by players.

Weber et al. argue that hierarchical systems are problematic solutions for RTS games because RTS games are non-hierarchical [8]. They require goals to be worked towards in parallel as there is usually three resources that must be balanced: resource aquisition, technology advancement, and unit production. Instead, Weber et al. propose an integrated agent framework where multiple specialized agents handle their designated portions of the game in parallel. It uses both a reactive planner (that adjusts plans based on the dynamic world) and an "external planner" which provides the overarching goal of a match (to win). While doing this, Weber et al. chose to limit the information provided to the system to the same information that a player would recieve. That is, the bot does not know where enemy units it can't see are, what resources an enemy has, or what technology level has been achieved, all in order to create a bot that plans and reacts in ways similar to a human player.

3 Approach

In order to handle the planning of multiple, complex objectives for a large number of computer-controlled units, we implemented a Hierarchical Task Network. First, it is important to discuss the environment provided by Arma 3. A "unit" is a single game object controllable by either a human player or the game's AI system. A unit is a single humanoid body and can range in appearance and gear from an equipped modern soldier to a guerilla fighter. Each unit belongs to one of four sides: BLUFOR, OPFOR, Independant, and Civilian. BLUFOR is "blue forces" or friendly forces, represented by the color blue. OPFOR (sometimes called REDFOR) is "opposing forces" or enemy forces, represented by the color red. Independent (sometimes called GREENFOR) is any military unit not belonging to wither BLUFOR or OPFOR, represented by the color green. Civilians are any unit considered to be a noncombatant. For this project, our HTN controls BLUFOR forces while fighting OPFOR units.

Units can be combined into a "group", a container of multiple units. A group acts as one and shares all information between its member units. The group is led by a single unit. This leader is the agent that calculates the micro plan for the group. If the leader is killed,

leadership is transferred to another unit in the squad. Groups typically range in size from a fire team (4 units) to a squad (8-13 units). Larger unit organizations are represented abstractly or not at all. We will refer to a group of groups as a "force", and will primarily work with groups controlled by a larger force.

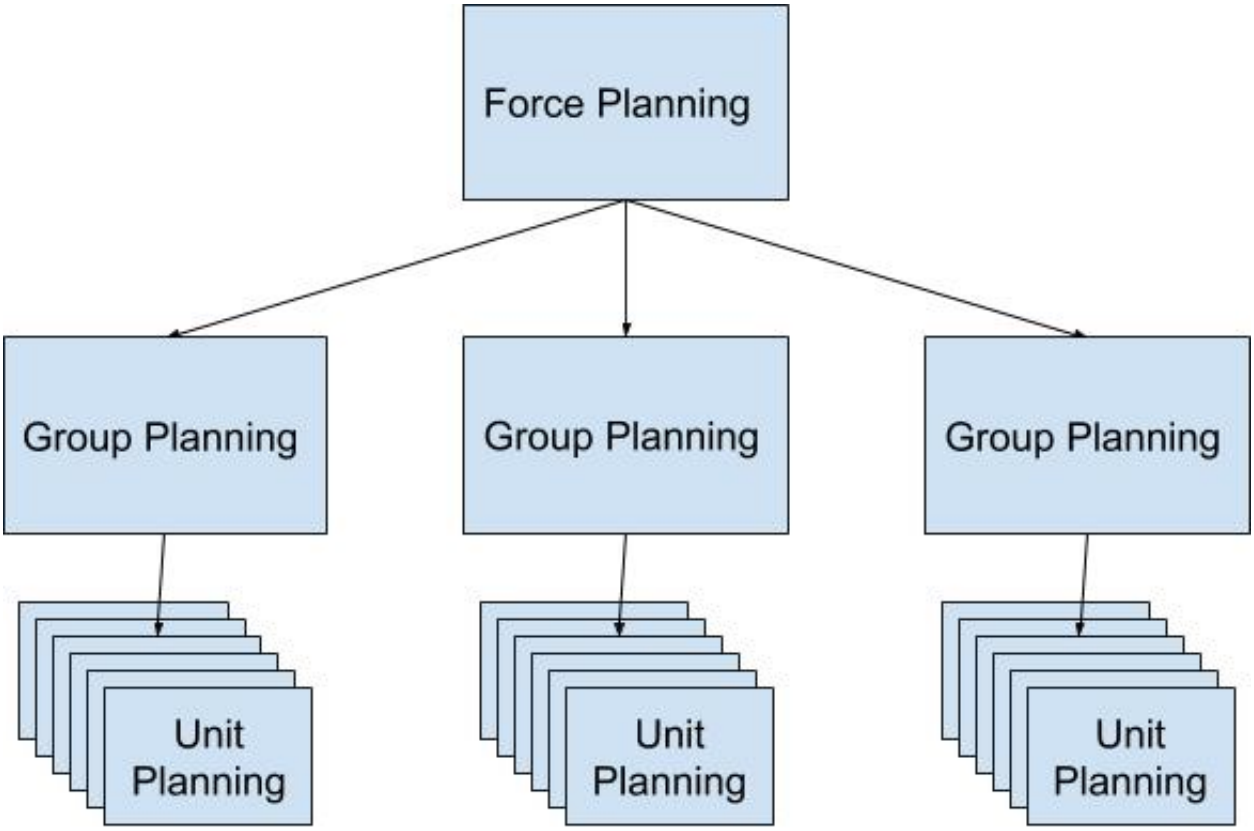

Figure 1: The hierarchy of tasks.

In order to provide a strategic plan for a force, there are three levels of tasks we need to deal with. This hierarchy of tasks is illustrated in Figure 1. Arma 3's AI comes shipped with its own micro planner. If an AI unit is spawned, it will sit motionless until given a direction. However, it will respond to attacks or aggressive actions from other AI unit or player units of another side. As the micro planner is already funtional, we only need to develop planning solutions for the group and force levels of the hierarchy.

The strength of a Hierarchical Task Network is the breaking down of abstract tasks into atomic actions. At the highest level, we have what we call "tasks". Tasks are ideas like "attack this squad", "defend this hill", or "search this area". Breaking it down a level, we have "waypoints". Waypoints are a data type native to Arma 3. They consist of a location and a type. We primarily use "MOVE", "SEARCH AND DESTROY" (attack), and "HOLD" (defend) waypoints. Following these waypoints is handled by the game's AI system, but simply placing a MOVE waypoint on an objective will cause a group to run straight towards it, ignoring maneuvering through defilade or other cover. We must treat

the placement of waypoints as pathfinding in order to produce realistic and desirable results.

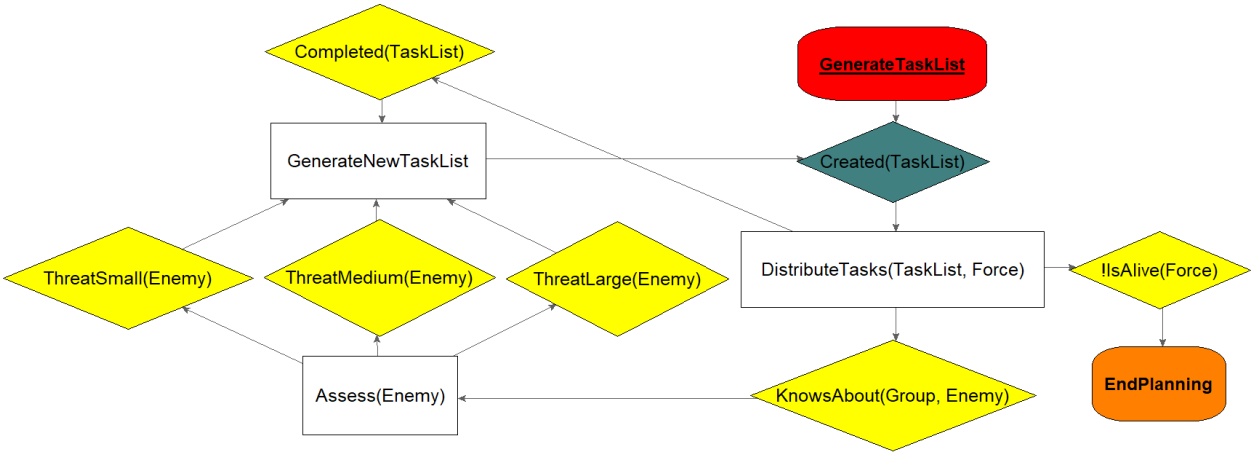

Figure 2: Force-level planning graph.

For the Force-level planning, the planning graph shown in Figure 2 was used. It takes a predefined list of objectives that need to be accomplished by the force. It uses these objectives to create a list of tasks that will accomplish them. Once the TaskList is created, it distributes those tasks to the groups in the force. It organizes the TaskList it sends to each group by a priority based on what each group should plan to accomplish. It decides this based off of each group's distance to the task destination, the strength of the group, the type and number of known or suspected enemies at the task location, and the danger level of the task.

Once each group in the force has it's tasks, the planner waits to modify or generate new tasks until one of three events occur.

If the entire force has been destroyed, planning is ended.

If every task in the TaskList is completed, then it will create a new defend task at the original objectives.

If new enemies are encountered by the groups, it will evaluate the threat level and develop new tasks accordingly. An enemy that poses a small threat will be completely ignored by the Force-level planner, as it will be handled by the Group-level planner. A medium threat will create a new task to destroy the threat and give it to a relevent squad. A large threat will create a task and delegate it to a proportional number of groups.

At the Group planning level, we use the planning graph shown in Figure 3. The goal of this planner is to turn the tasks generated by the Force-level planner into a list of waypoints that will accomplish each task.

The Group sees its provided TaskList as a stack. It creates a plan of waypoints for the first task in the TaskList, removing the task once it is complete. While executing the plan, a group will wait to re-evaluate its plan until one of several conditions are met.

Firstly, if it completes the plan, it will mark the current task as completed and then generate a plan for the next task. If there are no more tasks, it creates its own task to defend the group's current location until it recieves new tasks from the Force-level planner.

If all the units in the group have been killed, it will end the planning.

If the 3/4ths of the unit are killed, it will create a new task to retreat and then end the group's planning and remove itself from the force. If there is a nearby friendly group, a task will instead be created for the survivors to join the firendly group, then end the group's planning and remove itself from the force.

If an enemy is encountered, the group will immedietly pass that information to the Force-level planner. If the Force-level planner determines the enemy unit to be a small threat, then the Group-level planner will be left to determine its own plan to deal with the enemy. If the group has more units than the enemy group, then they will attempt to destroy it before continuing. If the unit begins losing the fight, it will attempt to retreat and request the Force-level planner delegate backup to its position. If the group determines that another unit is already tasked with attacking the enemy, it will ignore it unless otherwise directed by the Force-level planner.

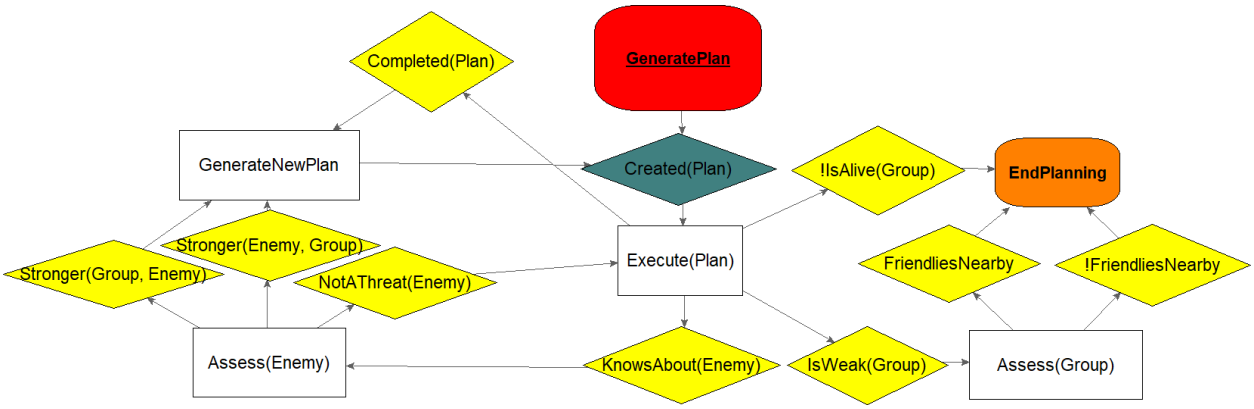

Figure 3: Group-level planning graph.

4 Experiment Design, Results, and Analysis

Because the goal of our Hierarchical Task Network is not to create unbeatable AI strategy, rather to automate the control of an AI-controlled enemy force in singleplayer or cooperative multiplayer experiences, coupled with the extremely long operating times of a standard Arma 3 mission (2-3 hours), our experimental design focuses on measuring the speed of plan generation. However, cursory feedback from playtesters gave generally positive feedback, primarily through the expression of interest in using our HTN to make mission creation easier. All tests are run on a Windows 10 PC with a 16 GB RAM, GTX 1070, and Intel i5-6600k.

We can measure the performance impact through the execution time of the *GenerateTaskList* and *GeneratePlan* functions. The search space for *GenerateTaskList* is simple, $f * t$ where f is the number of groups in the force and t is the number of tasks. The search space of *GeneratePlan* is more complicated since it includes finding a viable path from the group's

Table 1: Average Execution Time Results

Code	Avg. Execution Time (ms)	Execution Count
GenerateTaskList	0.023485	10000
GeneratePlan	0.252972	10000

Table 2: Average FPS Results

Mode	Avg. FPS	Length of Session (minutes)	Number of Sessions
Normal	47.54	20	5
x4 Speed	36.23	5	23

position to the task location. Since the pathfinding is not based on a grid, but instead plotting azthimuths towards or around terrain structure in 100 meter steps, the search space is dependant on the distance between start location and end locaton, as well as the number of obstacles encountered. In order to test the speed of planning execution, we called *GenerateTaskList* and *GeneratePlan* 10,000 times each and averaged their execution times. These results are shown in Table 1.

We can also measure performance by finding the average frames per second of the game over the course of a mission. A low server framerate will cause desynchronization issues between client and server, as well as negatively impact the micro behavior of AI-controlled groups, so maintaining a server framerate of at least 30 FPS is crucial. We monitored framrate over 5, 20 minute long sessions as well as 23, 5 minute long sessions at x4 game speed. The results are shown in Table 2.

Our final portion of the experiment is the most subjective and revolves around the visualization of the plans developed. Here we will demonstrate a generated plan for a mission consiting of two platoons, Alpha and Bravo, and their mission to attack two enemy-held bases. In this scenario, there are multiple patrols of enemies that the HTN will encounter and must deal with on the way to their primary objective. This can be seen in the Section 6: Appendix. The test here is to observe that the *GeneratePlan* function is creating realistic plans that follow terrain features and cover to the best of its ability.

Our HTN is modeled as a Finite State Machine. Arma 3 natively supports FSMs and since the default micro AI behavior is controlled via FSMs (and supports good performance while running hundreds of these FSMs simultaneously), the engine is biased towards processing FSMs extremely efficiently. Since tasks and plans are only generated when the HTN detects that a significant change has occured in the environment, the largest performance impact from our HTN will occur when a plan is generated. And because the area of operations of a typical Arma 3 mission stretches kilometers, the majority of any given mission is spent traveling. A plan may only be need to be calculated 20 times in a 2 hour mission. This all leads to a very lean planning process.

In a multiplayer environment, Arma 3's AI system is processed by the server. In a single-player environment, the player's computer acts as the server and handles the AI calculations. Because Arma 3 has infamously incompetent AI, typically an AI enemy force must number

5-8 times that of the players. It is common to see 300 enemy AI for 40 players in a cooperative mission. The primary reason for this is that the AI cannot flank, adjust their task on the fly, or even decide to attack in the first place. With the use of our HTN to organize and control AI units, we can reduce the number of AI units present in a cooperative mission, while still providing a challenge to players. This will save on server performance as there will be less AI to process, more than making up for the extremely slight cost of calculating tasks.

5 Conclusion and Future Work

In this paper we describe a solution for planning strategic action for large forces of AI units in the military-simulation video game Arma 3 using a Hierarchical Task Network. We took inspiration and advice from multiple publications on solutions for Real-Time Strategy bots that rival top human players. Our goal however was not to create an unbeatable AI opponent for Arma 3 players, but rather to provide the basis for a community tool that can be used to more quickly develop community missions and provide a higher degree of challenge to players. A secondary goal was to not tax the already expensive runtime environment of Arma 3. We believe our implementation will provide a necessary macro controller for AI strategy that makes mission creation faster and easier.

The current build of our HTN is focused solely on infantry versus infantry strategic planning. Obvious future work would be to write controllers for armored combat doctrine, air support, and transportation. We would also like to make a more robust intelligence system where the planner will task reconaissance groups with determining the location of enemy forces (but not engaging), as well as observing what an enemy unit or vehicle is equipped with and tasking an appropriately equipped group to deal with them (i.e. an enemy tank is observed and the nearest friendly group with anti-tank weapons is tasked with destroying it). We have created a solid and rudimentary base through which we can expand to cover a wide range of military doctrine simulation that was not possible to achieve during the time limit of this assignment.

6 Appendix: Plan Images

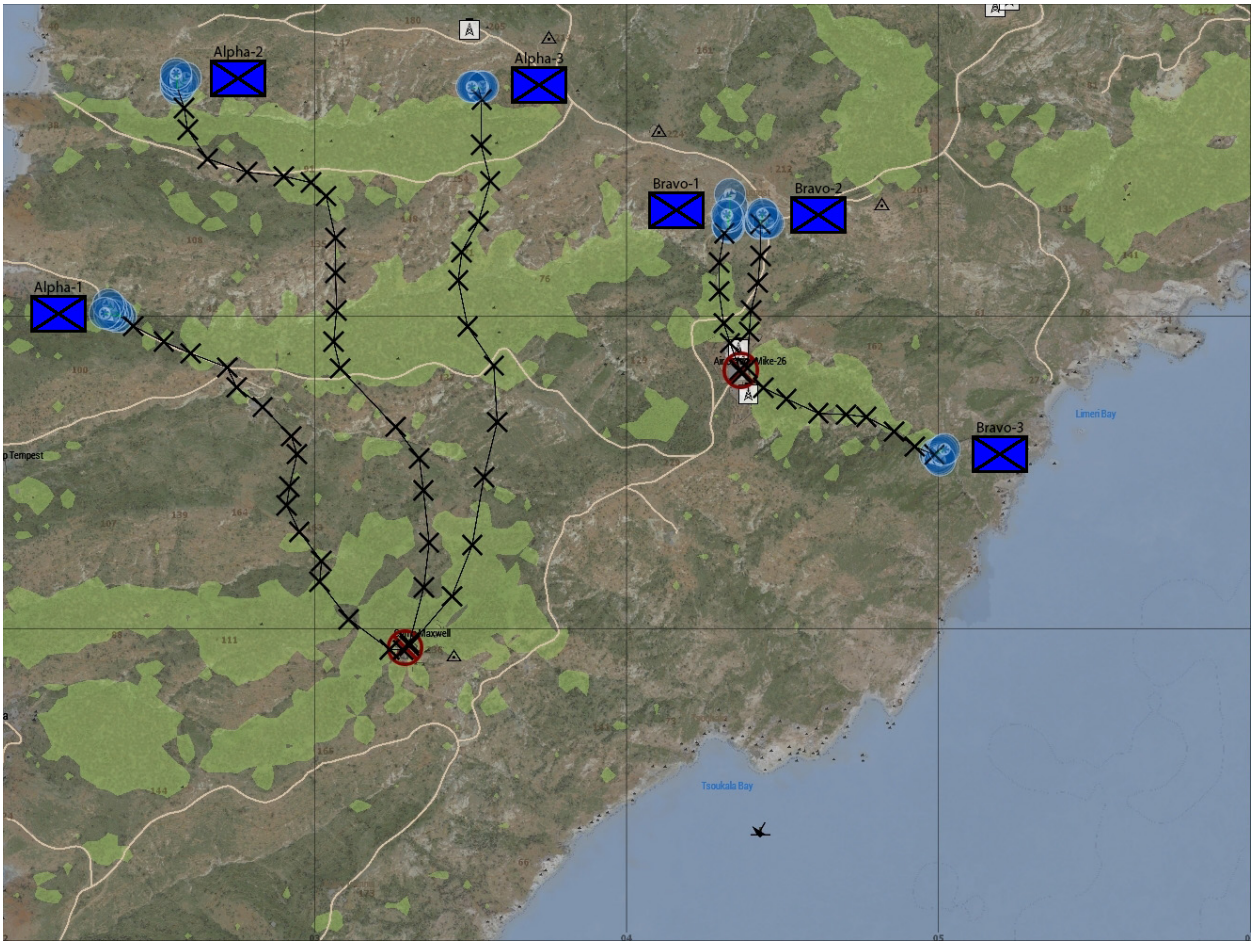

Figure 4: The initial plan generated by our HTN. Each 'X' is a waypoint or action it must perform to fulfill its generated plan.

Figure 5: Close up of Alpha-1, Alpha-2, and Alpha-3 groups and their plans.

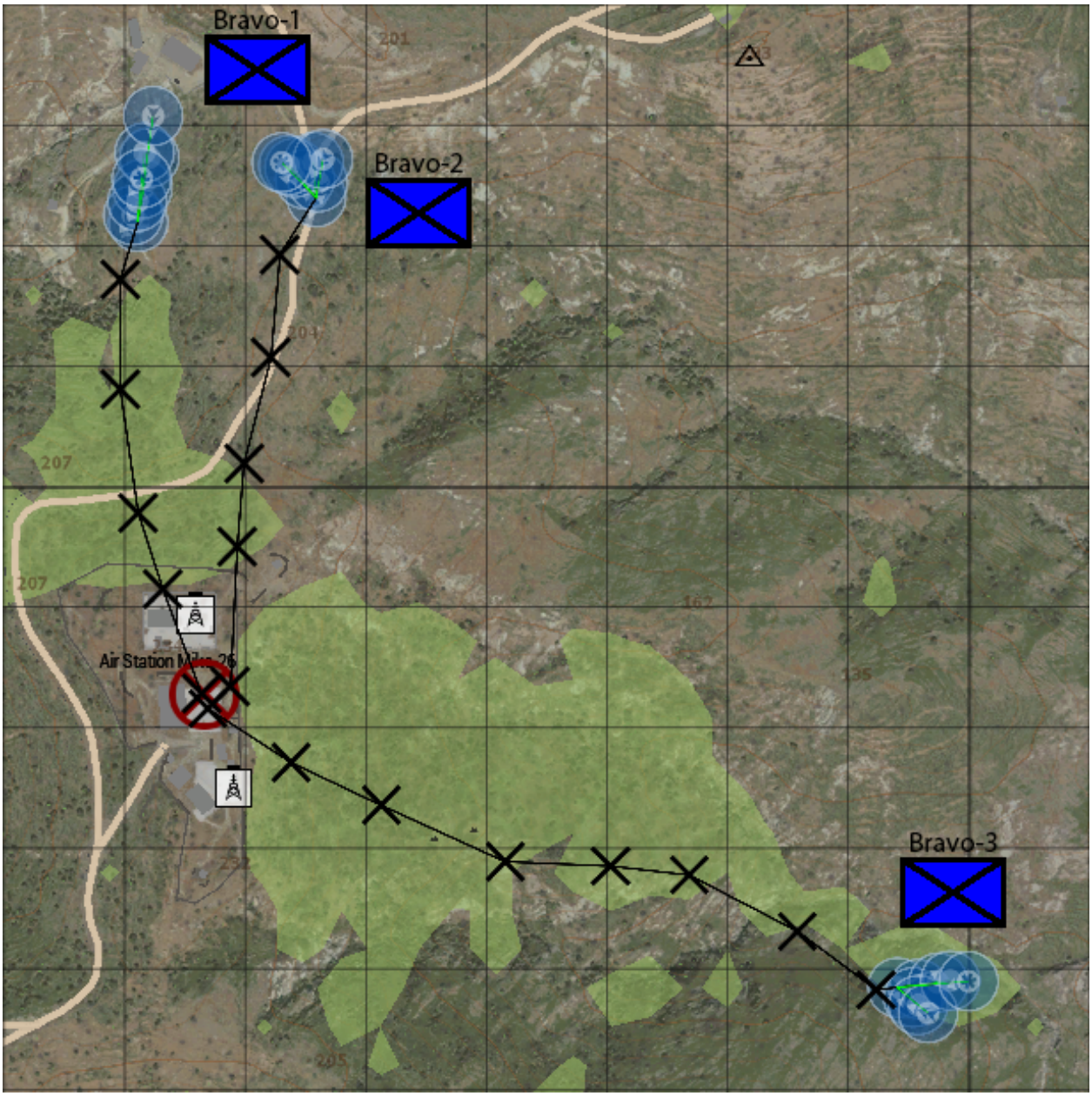

Figure 6: Close up of Bravo-1, Bravo-2, and Bravo-3 groups and their plans.

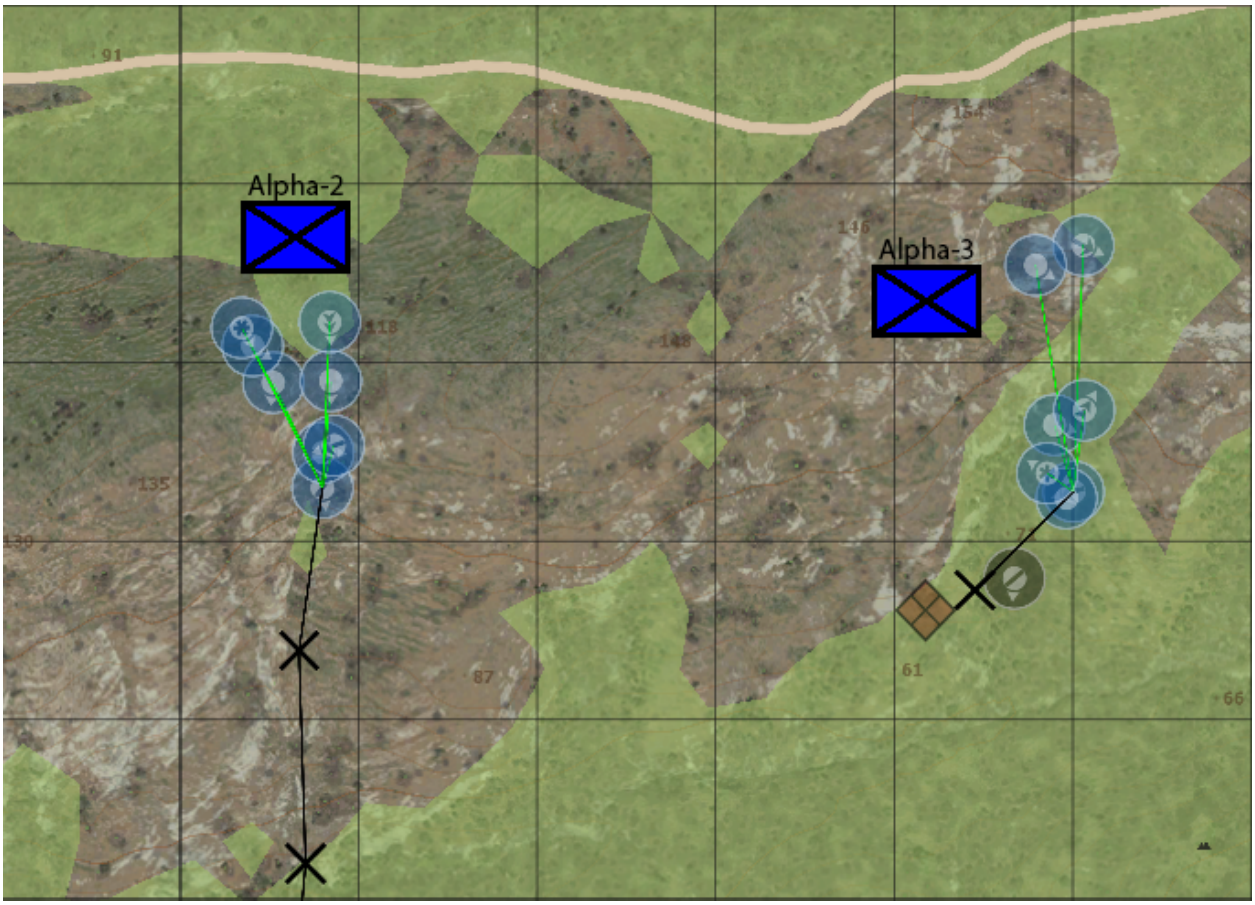

Figure 7: Alpha-3 encounters enemies (red diamond) and is assigned a task to destroy them. Alpha-3 generates a plan to destroy the enemy. Alpha-2 also detects the enemy, but because Alpha-3 has already been tasked, Alpha-2 does not change its original plan.

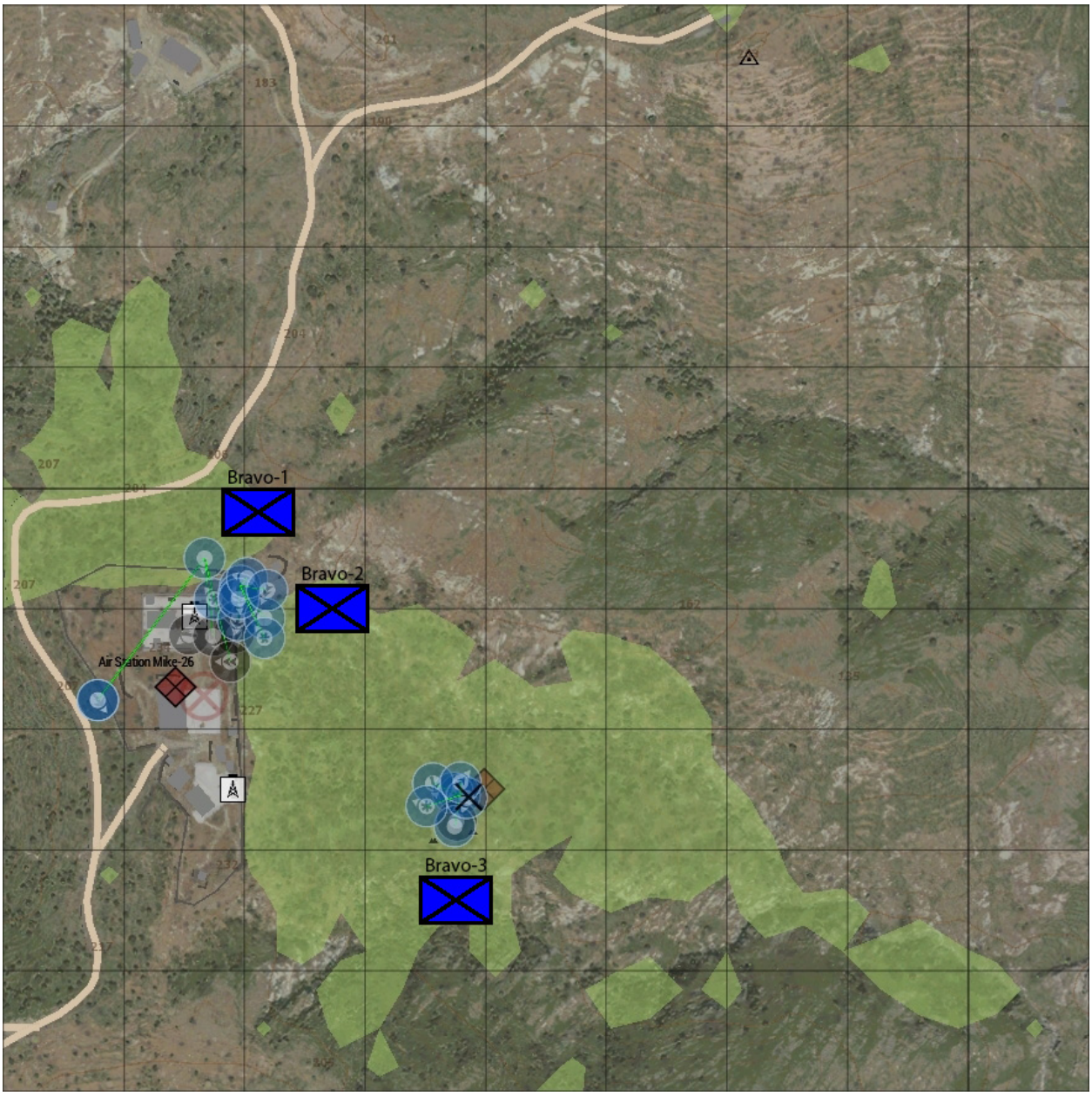

Figure 8: Likewise, Bravo-3 encounters enemies on it's way to it's objective. It is assigned a task to destroy them. Bravo-1 and Bravo-2 encounter enemies at their objective and so are assigned the task of destroying them.

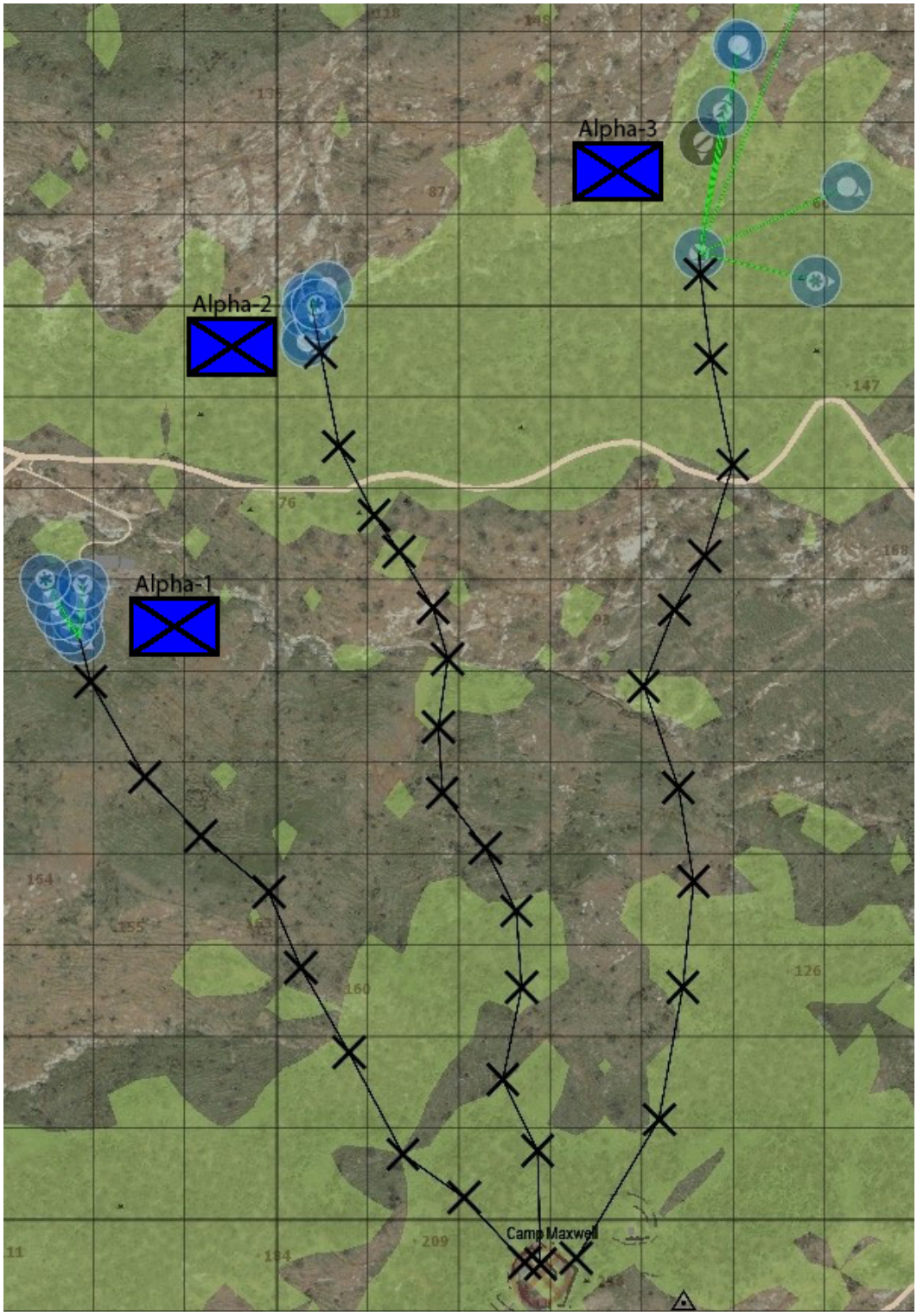

Figure 9: Alpha-3 succeeds in destroying the enemy squad and resumes its original task, generating a fresh plan. The group did lose their squad leader (grey icon) and so a new leader is promoted (the autorifleman). The leader is indicated by green lines connecting each unit to the leader unit.

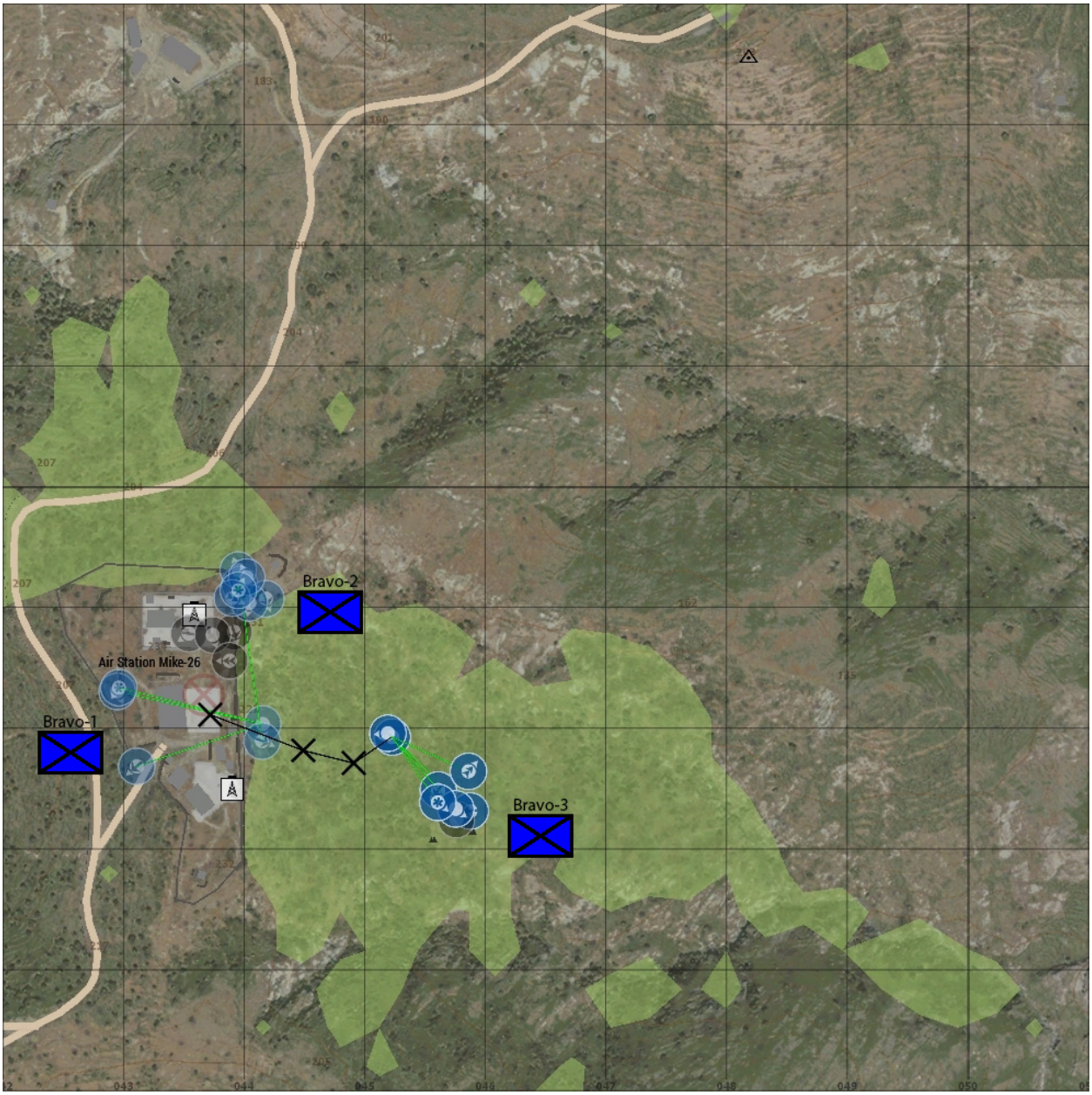

Figure 10: Bravo-3 also destroys it's enemy contact and resumes its original task, generating a fresh plan. Bravo-1 and Bravo-2 are still engaged with the enemy.

Figure 11: Alpha Platoon has made its way to its task, with Alpha-3 being assigned a task to destroy newly encountered enemies. Bravo platoon has finished its first task now and gets assigned the same task as Alpha. Bravo-1 and Bravo-3 generate fresh plans for Alpha's task, while Bravo-2 has been assigned to attack the enemies Alpha-3 encountered.

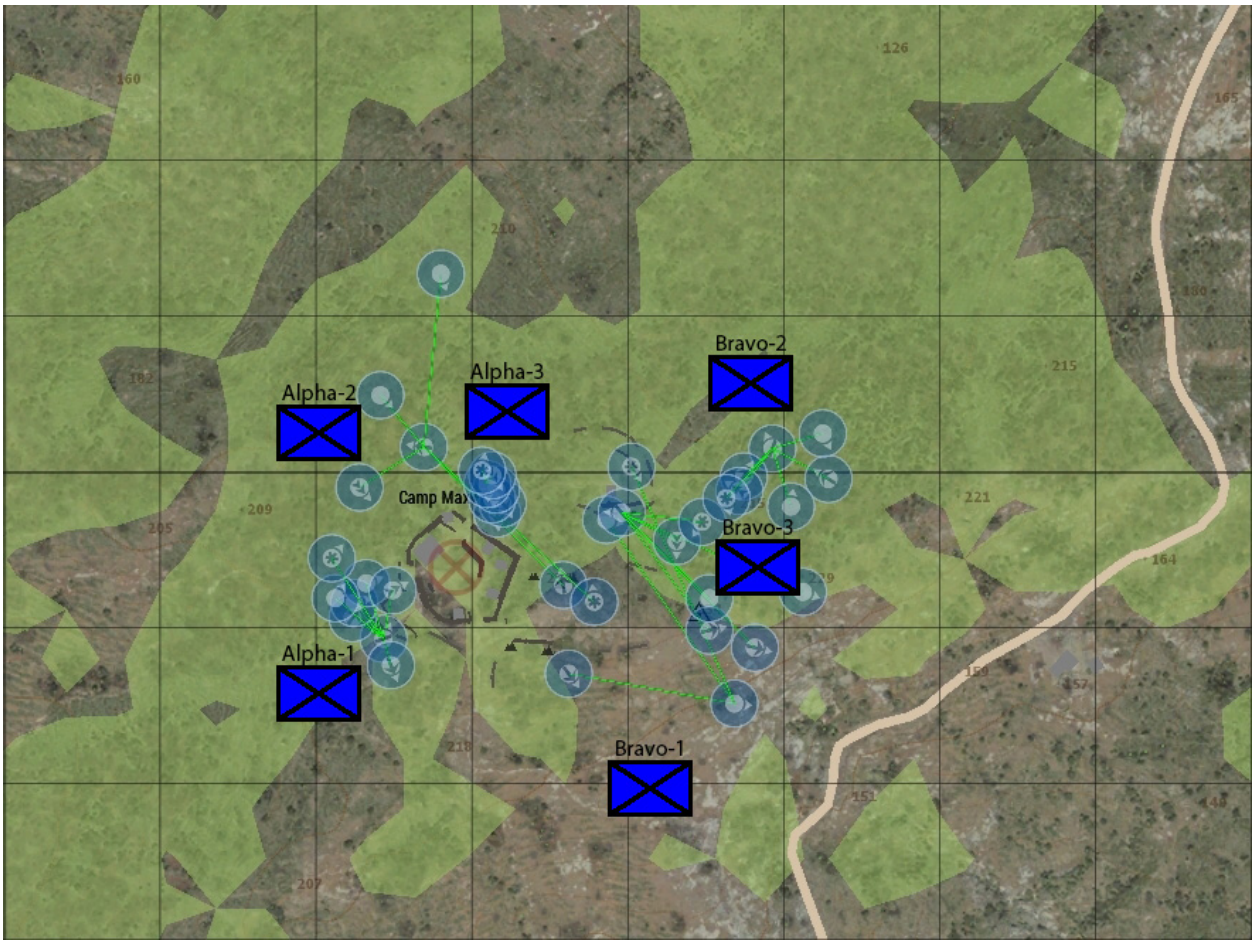

Figure 12: Every task has been completed and all encountered enemies have been destroyed. Alpha and Bravo are tasked to remain at their positions until new tasks are assigned. The mission is over.

References

[1] J. Baxter and R. Hepplewhite. A hierarchical distributed planning framework for simulated battlefield entities. 2001.

[2] P. Bjarnolf, P. Gustavsson, C. Brax, and M. Fredin. Threat analysis using goal-oriented action planning. *FallSIW*, 2008.

[3] P. Gorniak and I. Davis. Hierarchical planning and coordinated plan execution for squads of characters. *AIIDE*, 2007.

[4] H. Hoang, S. Lee-Urban, and H. Munoz-Avila. Hierarchical plan representations for encoding strategic game ai. *AIIDE*, 2005.

[5] S. Ontanon, K. Mishra, N. Sugandh, and A. Ram. Case-based planning and execution for real-time strategy games. *ICCBR*, 2007.

[6] S. Smith and D. Nau. Strategic planning for imperfect-information games. *Institute of Systems Research*, 1993.

[7] G. Synnaeve. Bayesian programming and learning for multi-player video games. 2011.

[8] B. Weber, M. Mateas, and A. Jhala. Building human-level ai for real-time strategy games. *AAAI*, 2011.