TRAC-M-TR-13-054
15 JUL 13

Knowledge Representation for Decision Making Agents

TRADOC Analysis Center - Monterey
700 Dyer Road
Monterey, California 93943

This study cost the
Department of Defense approximately
$27,000 expended by TRAC in
Fiscal Years 12-13.
Prepared on 20130715
TRAC Project Code # 643

This page intentionally left blank.

TRAC-M-TR-13-054
15 JUL 13

Knowledge Representation for Decision Making Agents

MAJ Peter Nesbitt
Dr. Tom Anderson
LTC Jonathan Alt
Mr. David Ohmen
Mr. Kyle Quinnell
Mr. Mario Torres

TRADOC Analysis Center - Monterey
700 Dyer Road
Monterey, California 93943

PREPARED BY:	APPROVED BY:
Peter A. Nesbitt MAJ, AR Analyst, TRAC-MTRY	Jonathan K. Alt LTC, IN Director, TRAC-MTRY

This page intentionally left blank.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 07/15/2013	**2. REPORT TYPE** Technical Report	**3. DATES COVERED** *(From - To)* APR 2013 - JUN 2013

4. TITLE AND SUBTITLE Knowledge Representation for Decision Making Agents	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) MAJ Peter Nesbitt Dr Tom Anderson LTC Jonathan Alt Mr. Daid Ohmen Mr. Kyle Quinnell Mr. Mario Torres	**5d. PROJECT NUMBER** 643
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) TRADOC Research Analysis Center, Monterey, CA Naval Postgraduate School, Monterey, CA	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) TRADOC Research Analysis Center, White Sands Missile Range, NM	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) TRAC-M-TR-13-054

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This technical memorandum documents the implementation of using agent knowledge to inform dynamic behaviors within COMBATXXI. This project partially addresses TRAC research requirements 4.1, 4.5, 4.9, 4.10, 4.13, 4.18, and 4.19 under Support for TRAC M&S and Scenario Enterprise. The results are a methodology that can be used to expand the agility and responsiveness of COMBATXXI for future study use. We demonstrate it is possible to implement non-prescriptive dynamic behavior through use of a knowledge map without modification to the COMBATXXI base code. We also show there are several secondary benefits to this methodology, including representing collective awareness.

15. SUBJECT TERMS
knowledge map, COMBATXXI, dynamic behavior, modular scenario development, dynamic decision making, situational awareness, threat, HTN, simulation, combat model

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Peter Nesbitt
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	SAR	60	(831) 656-7575

This page intentionally left blank.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This technical memorandum investigates insights gained for future use of knowledge maps informing dynamic behavior in the Combined Arms Analysis Tool of the 21st Century (COMBATXXI). Current methods of knowledge representation are brittle applications of dynamic decision making. The capability to model robust agent behavior will reduce the time required by scenario developers, potentially improving the agility of simulation models, and provide explainable and traceable agent level decision making.

This project partly addresses TRAC research requirements 4.1, 4.5, 4.9, 4.10, 4.13, 4.18, and 4.19 under Support for TRAC Modeling and Simulation and Scenario Enterprise [2]. This work, which focuses on developing complex decision making within the current release version of COMBATXXI, is in line with TRAC research priorities.

Currently, decision making agents generally depend on immediate sensing to determine the state of the simulation environment. This often short sighted, first hand sensing supports limited courses of action development and selection by a decision making agent.

The approach investigated by this project overcomes this limitation by developing a means to synthesize and store pertinent information representing knowledge. The technical approach includes using Python script in the development of a knowledge map to inform behavior in COMBATXXI. This approach is intended to enable autonomous, but explainable, behavior and potentially reduce the time required to “hard-wire” behaviors within COMBATXXI. This approach will allow non-prescriptive dynamic behavior as well as a representation of collective knowledge. Integrating a knowledge map into COMBATXXI is accomplished through BSL scripting language calls in COMBATXXI to supporting Python scripts. One script controls the knowledge map, collecting and processing agent knowledge into a common picture. Another script serves as the behavior control defining how the agent processes the knowledge map in order to determine a course of action.

We demonstrate it is possible to implement non-prescriptive dynamic behavior through use of a knowledge map without modification to the COMBATXXI base code. We also show there are several secondary benefits to this methodology, including representing collective awareness. The application of this new technique is applied to the COMBATXXI Deployed Force Protection (DFP) scenario, successfully demonstrating complex decision making. Future work is required to identify how the behavior affects runtime.

LIST OF ACRONYMS AND ABBREVIATIONS

BSL	Behavior Scripting Language
COMBATXXI	Combined Arms Analysis Tool of the 21st Century
DFP	deployed force protection (scenario)
HTN	hierarchical task network
IDE	integrated development environment
IPR	interim progress report
MCCDC	Marine Corps Combat Development Command
MTRY	Monterey
METT-TC	mission, enemy, time, terrain - troops, civil considerations
MGRS	military grid reference system
NAI	named area of interest
SITS	Scenario Integration Tool Suite
TRAC	TRADOC Analysis Center
TRADOC	Training and Doctrine Command
WSMR	White Sands Missile Range

1. INTRODUCTION

Combat models, including Combined Arms Analysis Tool of the 21st Century (COMBATXXI) [3], simulate military operations in order to answer analysis questions. This technical memorandum investigates insights gained for future use of knowledge maps informing dynamic behavior in COMBATXXI. Current methods of knowledge representation are brittle applications of dynamic decision making. The capability to model robust agent behavior will reduce time required by scenario developers. This can potentially improve the agility of simulation models, and provide explainable and traceable agent level decision making and may contribute greatly to answering analysis questions. The techniques explored in this proof of concept may prove valuable to a methodology to mass produce scenarios that inform on increased study space and maintain doctrinal integrity. All data, scenarios, and documentation referenced in this report is located under project 643 in the TRAC Knowledge Management System at the address trac/Projects/643/.

1.1. Problem Description

The need exists to represent dynamic behavior for decision making agents in combat simulations. This project partly addresses TRAC research requirements 4.1, 4.5, 4.9, 4.10, 4.13, 4.18, and 4.19 under Support for TRAC M&S and Scenario Enterprise. [2] Currently, COMBATXXI can model decision making agents in many ways including in the SITS Orders Behavior Tables 2.3 or in a supporting behavior module such as an hierarchical task network editor. These methods rely on conditional rules that use only immediately sensed influenced to identify the appropriate behavior rule. Processing and organizing sensed information will improve the current means to control behavior. This project focuses on developing capabilities in representing knowledge in support of complex decision making agents using the Stablebuild 20130514 version of COMBATXXI [3].

1.2. Scope

Problem Statement. This project demonstrates a proof of concept use of graphs to represent knowledge enabling dynamic behaviors within COMBATXXI.

Constraints. Constraints limit the team's options to conduct the project [16].

- Demonstrate complex decision making with knowledge maps within the current release version of COMBATXXI and must not rely on modification to the source code.
- Software development support was constrained to two TRAC-WSMR developers for the month of April 2013.

Limitations. Limitations are a team's inability to investigate issues within the sponsor's bounds.

- Verification of new techniques is limited to the DFP scenario and addressing UAS control.
- The demonstrated capability is applicable to existing scenarios.

Assumptions. Assumptions are specific statements that are taken as true in the absence of facts.

- Behaviors will integrate with COMBATXXI general use to enable autonomous decision making for a variety of use cases.
- The specific military situation of a UAS (representing an operator and aerial vehicle) simulating a search for a target adequately demonstrates the utility of a knowledge map. This situation is leveraged throughout this report.
- The methods demonstrating UAS control can be generally applied to controlling other military systems.

1.3. Methodology

An incremental approach was used, first demonstrating the concept in Python alone and then using this Python demonstrator to guide the development of a Python behavior in COMBATXXI. Finally, the method was applied to a specific COMBATXXI scenario. The following list is the project's approach to the problem. Figure 1.1 graphically shows the methodology for this project.

1. Define the problem.
2. Identify potential solution for prototyping.
3. Develop Python proof of principle including:
    - UAS target search scenario.
    - Network based representation of knowledge.
    - Action selection using greedy algorithm.
4. TRAC-WSMR implements knowledge map based search behavior with UAS into COMBATXXI.
    - Integrate COMBATXXI and knowledge map script.
    - Implement knowledge map informing agent behavior.

- Update knowledge map according to new agent sensing.

5. Apply technique to DFP scenario to demonstrate a proof of concept.

6. Verification.

7. Complete Technical Memorandum.

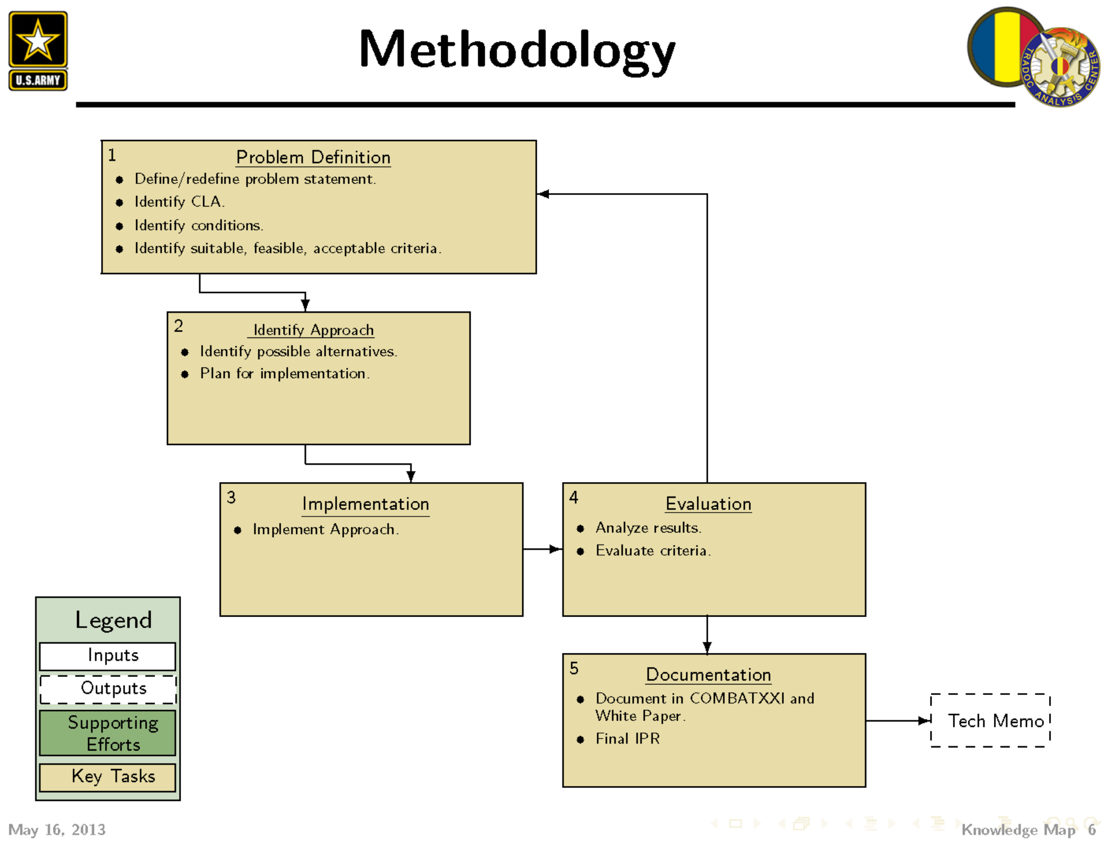


Figure 1.1: Methodology flowchart for Project 643, Knowledge Representation for Decision Making Agents from initial IPR brief, 15 March 2013 [12].

This page intentionally left blank.

2. BACKGROUND

Knowledge representation is the focus of this technical memorandum. We consider knowledge representation as the means to which information is collected, processed and shared for the purpose of making a decision. In a military scenario, an agent's behavior is dependent on its perceived state of the world. In this chapter, we discuss current and proposed methods determining agent actions.

2.1. COMBATXXI

COMBATXXI is a combat model owned by and currently used to support analysis in TRAC. It is a high-resolution, stochastic simulation used for analysis generally at brigade and below. As a closed form simulation, it requires a means of decision-making and behavior that replicate the effects of combat without a human in the loop. [13]. The current approach of behavior control in COMBATXXI is very capable of controlling single entities based on what it senses. Current techniques also allow one entity to sense an event and communicate orders to one or many other entities to act. Decision making agents generally depend on immediate sensing to determine the current state of the simulation environment. This often short sighted, first hand sensing supports limited courses of action development and selection by a decision making agent. Dynamic behavior techniques allow for better case handling, however these techniques require a means to categorize a situation in order to select an appropriate behavior.

2.2. Explicit Script Control

One basic method of developing a scenario is to explicitly script every action of the agent. This method is common and tightly controls every agent's behavior to a predetermined script, see Figure 2.1. Every action is determined and can be laborious by the scenario developer. UAS air routes created with this method can be seen in Figure 2.2.

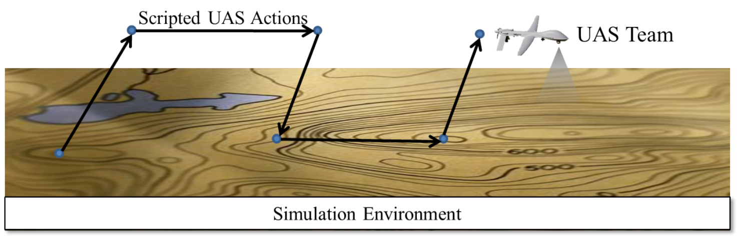


Figure 2.1: Concept sketch including a single UAS agent simulating a scripted search for a target. As it searches, the effects of sensors do not inform a knowledge map or any change in behavior.

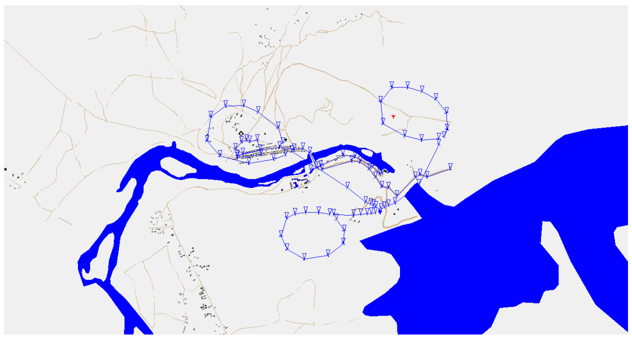


Figure 2.2: Example of scripted control in COMBATXXI. The three air routes of waypoints are used to explicitly control the a single UAS search route [11].

2.3. Behavior in COMBATXXI

A COMBATXXI behavior is a set of instructions that simulates the decision making process. These instructions map an agent's perceived state to next necessary action demonstrated in Figure 2.3. Behaviors are similar to what Soldiers call tactics, techniques and procedures (TTPs). Behaviors operate at different levels: Cognitive, Tactical, Procedural, and Primitive. COMBATXXI Behavior Development Tools include Behavior Specification Language (BSL), Python (Jython), Cognitive Behaviors for Units and Soldiers (CBUS), Behavior Tables, and Maneuver & Fires Tools [9].

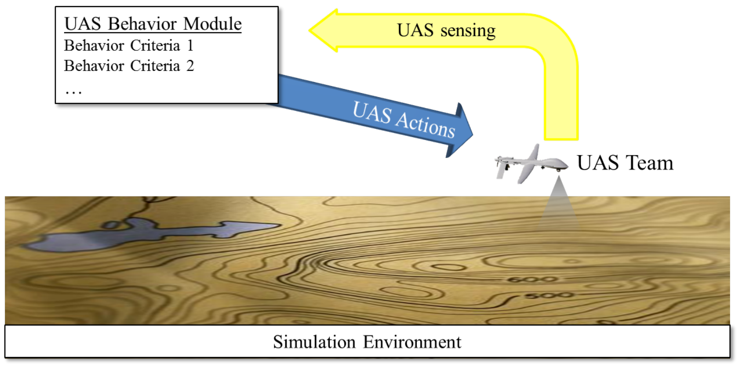


Figure 2.3: Concept sketch including a single UAS agent simulating a dynamic search for a target. A behavior control module assesses only the immediate available information and selects a next location to search.

Following are a few tools available that build behaviors in COMBATXXI.

- SITS IDE - The Scenario Integration Tool Suite (SITS) has a Behavior Properties Editor enabling the development of behavior criteria for decision making agents defining force structures, unit organizations, entity configurations and operational details. [15]

- HTN Editor - Hierarchal Task Network organizes the dependencies and sequence of mapping states of the simulated environment to directed actions of the decision making agent in the form of networks or trees.

The current behavior methods can benefit from knowledge representation methods that allow for decision making agents to synthesize disparate sources of information in the scenario and maintain a flexible representation of the current state for the application of their behavior criteria. This approach is intended to enable autonomous, but explainable, behavior and potentially reduce the time required to “hard-wire” behaviors within COMBATXXI.

2.4. Knowledge Representation

Current behavior control methods generally have access only to immediate sensing to determine the current state of the simulation environment. This method of representing knowledge is brittle and has difficulty accounting for situations not specifically considered by the developer of the scenario due to a limited means to assess the options in the context of other important information in the scenario. The decision is essentially conducted in a memoryless condition depending only on available information about the current time and not on information from further in the past. We saw an example of this in Figure 2.3. Generally, it is necessary to explicitly control every scenario developed so the the range of possible courses of action by a decision making agent are covered in a set of state-to-action cases in SITS Orders Behavior Tables [15] or handled by a branch in an HTN. To avoid states for which there is no direct mapping and mitigate invalid behavior, strict control is maintained for each agent through control measures and transition of goals. This strict control is potentially laborious.

There are many benefits to improving knowledge representation by using knowledge maps to inform behaviors. This dynamic mapping from assessment of a perceived state to an available action allows for the following:

1. *The extension of limited scenario resources* may be achieved through use of behavior modules informed by knowledge map modules. These modules are created to describe how an agent understands its environment, behaves, and communicates regardless of scenario and are applied across multiple environments and studies. The capability to build behaviors with modules transfers development time away from prescribed micro movements and communications, reducing scripting time. Using these approved and appropriate building blocks to create battlefield geometries necessary for study scenarios may not only simplify the scenario construction process, but also serve to be a clear translation from mission intent graphics to execution.

2. *Behavior consistency* is maintained across differing situations. The capability to build behaviors as modules allows for approval and verification of one behavior

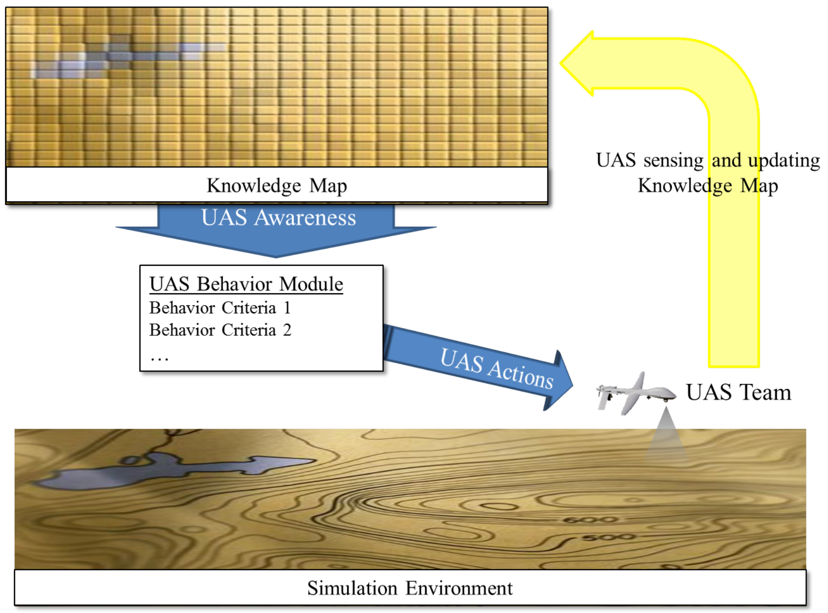


Figure 2.4: Concept sketch including a single UAS agent simulating a search for a target. As it searches, the effects of sensors inform a single knowledge map. A behavior control module assesses the information in the knowledge map, as well as other sources (such as current location of the agent) and selects a next location to search separately for each agent. This single knowledge map application allows for collective behavior coordination.

module for application in many different scenarios as seen in Figure 2.5. This robust implementation of intended agent's behavior is achieved through implicit application of behavior criteria. An agent transitions between approved actions in accordance with its assigned modules as the situation and transition criteria dictates.

3. *Persistence of previously sensed information* in a graph allows knowledge of the past to be carried forward in time for use in later decisions. A diagram of a decision making agent in contact with a source of information and collecting it as knowledge for the application of its rules of behavior is demonstrated in Figure 2.4.

4. *Appropriately complex* representation of decision making agents. An agent's behavior can be quickly modified through a behavior module swap to meet the needs of the study.

5. *Collective awareness* is the representation of the collected and processed information from a set of agents updated during the simulation. Decision making agents in contact with sources of information not directly sensed by the agent will use these sources in the application of its rules of behavior demonstrated in Figure 2.6. The capability for representation of group awareness to inform individual agent's decisions can contribute in the support of a study in several ways and allows for the following:

   - *Networked behavior* augments the quantity and quality of information available to a decision making agent in assessing the state of the simulated environment.

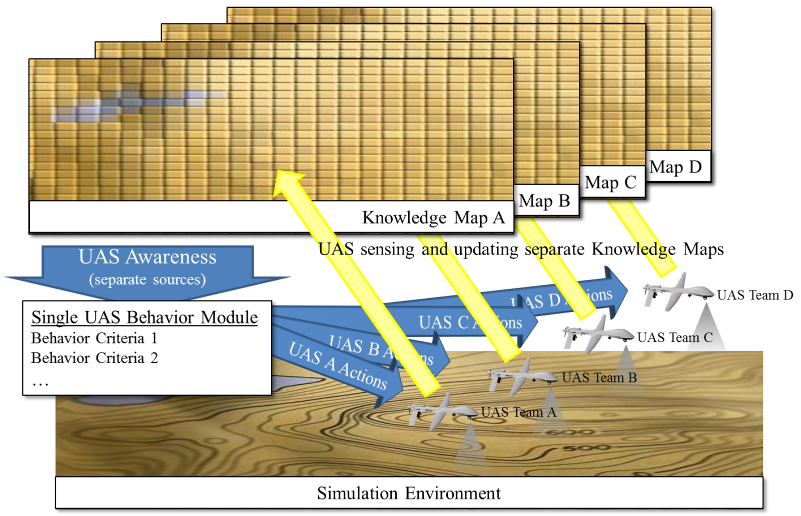


Figure 2.5: Concept sketch including a multiple UAS agents simulating searches for targets. As each UAS agent searches the effects of its sensors its own knowledge map. A single UAS behavior control module assesses the information of each particular knowledge map and selects a next location to search separately for each agent. This multiple knowledge map application requires a single UAS behavior, and multiple agents with their its own copy of one type of knowledge map.

The ability to converge disparate reports and create a state of communal knowledge allows a capability to answer and execute the following behaviors. It is possible to allow many agents to sense individually inconsequential events and combine the information into a knowledge map. The combination of the many disparate reports builds a more complete picture of the perfect knowledge state. Accessing the synthesized information in the knowledge map allows agents to make a decision based on the group's understanding of the current situation.

- *Coordination of effort* is dynamically achieved. Decision making agents can act within the restrictions and guidelines of a group. The simulation can account for the individuals acting in a coordinated effort of the group across space and time. This achieves the effect of a group of UAS controllers sharing information in a single room, coordinating their search. Also, if important information is not obtained by any contributer to the knowledge map, it is not available in the knowledge map.

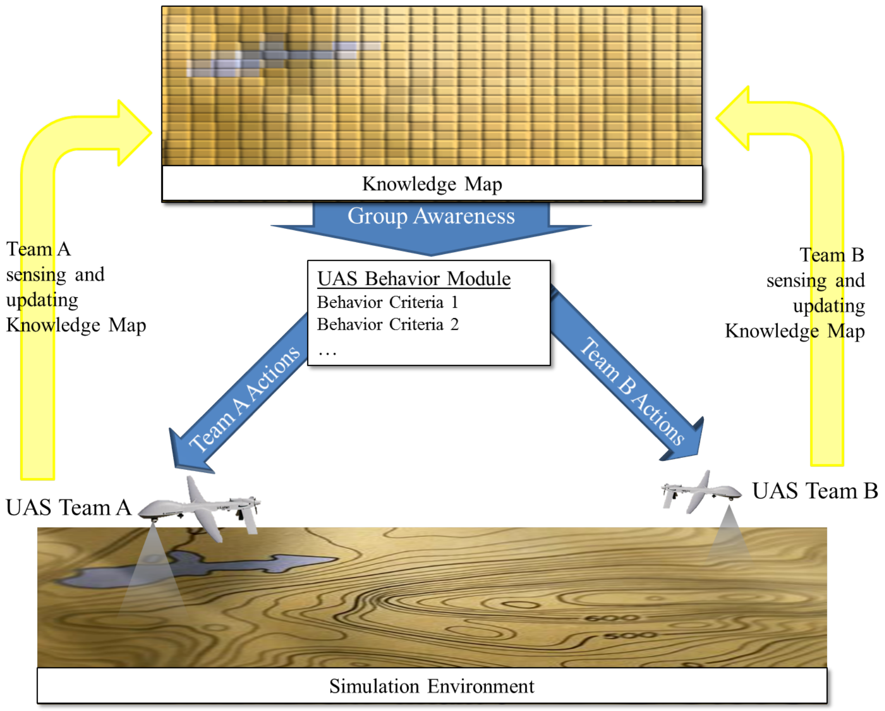


Figure 2.6: Concept sketch including two UAS agents simulating a search for a target. As they search, the effects of both agents inform a single knowledge map. A single or multiple behavior control modules assesses the information in the knowledge map, as well as other sources (such as current location of the agent) and selects a next location to search separately for each agent. This single knowledge map application allows for collective behavior coordination.

3. TECHNICAL APPROACH

This technical report investigates the development of a knowledge map to inform behavior control and improve dynamic behavior in COMBATXXI. In order to accomplish this, there is a requirement for a means to collect and store pertinent information in the scenario to aid in decision making. A knowledge map in the form of a graph will accounts for and represent changes in an agent's perceived knowledge giving the effect of situational awareness. The behavior control methods then have access to this important information as knowledge to make more informed decisions and possibly allow for an extensive application of dynamic behaviors.

3.1. Knowledge Map as a Graph

Simulation agents access the knowledge map maintained in a graph in order inform a behavior and select a specific action available for the given situation. We define our graph (also called a network) as a collection of nodes and edges.

$n \in N$	node n representing a location in set of all locations N.
$e \in E$	edge e representing a location in set of all edges E.

Table 3.1: Table of indices and sets for knowledge map as a graph.

Associating information to each node allows information to be processed, stored and persist as knowledge for use in later decisions. In our demonstration, there is a value for the likelihood of enemy presence at each node shown in Table 3.2.

$\gamma_{(n)}$	coordinates for location of node n in simulation. [MGRS]
$\lambda_{(n)}$	likelihood of enemy presence at node n. [0,1]

Table 3.2: Table of data for knowledge map as a graph.

We will demonstrate the method of action selection with a simple greedy algorithm. Figure 3.1 shows our simple behavior that selects a next way point for our agent.

```
Define GreedyAlgorithm($M$):
    maxLikely = 0
    for every $n \in N$,
        if $\lambda_{(n)} >$ maxLikely:
            maxLikely = $\lambda_{(n)}$
            waypoint = $\gamma_{(n)}$
    return waypoint
```

Figure 3.1: Pseudo code for greedy algorithm selecting next waypoint using definitions from Table 3.1 and 3.2 .

For our demonstration, the knowledge map M = *(N,E)* is a finite, fully connected, directed graph. Even though nodes can be added to the graph, removed or otherwise modified by aggregation or partitioning, we do not change the structure of the graph after instantiation[1]. Changing $\lambda_{(n)}$ reflects changes in the environment from the perspective of the agent such as effects of sensors, speed of evasive enemy, etc. The behavior selects the next waypoint as the location with the most obvious immediate advantage by maximizing the likelihood of enemy presence. At this location $\lambda_{(n)}$ is updated with the effects of the information gained through sensing. This process continues until an event changes the behavior such as finding the target, low fuel or destruction of the agent. Figure 2.4 is a concept sketch demonstrating how the agent (UAS team) senses and updates a knowledge map.

4. APPLICATION

This report demonstrates the use of a knowledge map as a graph and behavior control in the scenario of simulating dynamic search behavior in first Python, then COMBATXXI. The use of a knowledge map is not limited to this demonstration scenario or combat model.

4.1. Application in Python

TRAC-MTRY developed Python scripts for knowledge map demonstration titled `virtualCBTXXI_v2.4.py` with code found in APPENDIX A, `greedyModule.py` found in APPENDIX B, and `myopicGreedyModule.py` found in APPENDIX C. The script `virtualCBTXXI_v2.4.py` contains a knowledge map that accounts for and represents the UAS's understanding of enemy presence. This simple knowledge map in the form of a Python dictionary data structure is a collection of locations identified by a x,y coordinate and the associated belief of enemy presence at that location. The scripts `greedyModule.py` and `myopicGreedyModule.py` are separate behavior control methods that leverage the knowledge representation. The basic pseudo code for the process of updating and accessing the knowledge map is as follows.

1. **Instantiate knowledge map.**
2. **Check stopping criteria.**
3. **Select course of action using a knowledge map informing behavior.**
4. **Execute behavior.**
5. **Sense the environment.**
6. **Update knowledge map.**
7. **Repeat steps 2 through 6.**

Using the steps above, the following procedure further details the order of events during the search.

1. **Instantiate knowledge map.** This knowledge map is a dictionary data structure called `tmap` in the code. It represents a network of locations with a number [0,1] representing the UAS perception of likelihood of enemy presence at that location. This number is modified by the effects of initial belief, terrain, sensors, enemy templates or reports. The demonstrator allows for one of three ways to initially fill these attribute values, controlled by the variable `initialBelief`.

   - `initialBelief` = 1 uses `fillUniform()`: Call to fill all cells with same uniform probability (Random Walk).

- `initialBelief` = 2 uses `fillRandom()`: Informed initial tmap distribution (randomly generated per node) with belief one.
- `initialBelief` = 3 uses `fillCenter()`: normal distribution centered on the network.
- `initialBelief` = 4 uses `fillSin()`: sin curve with apex centered on network.

2. **Check stopping criteria.** Validation of stopping criteria is checked before a new search location is assessed. This code includes three stopping criteria: limit on distance traveled, limit on glimpses (sensor uses) and an acceptable tolerance on believed lack of enemy presence. If any stopping criteria are met, the search is complete.

3. **Select course of action using a knowledge map informing behavior.** If a stopping criteria is not met, a new location is selected for the continued search. Two modules demonstrate how a UAS agent could use the knowledge map to select a location to fly to next. These modules are `greedyModule.py` and `myopicGreedyModule.py`.

   - `greedyModule.py` -guides the UAS to fly to the location anywhere in the search area with the greatest likelihood of enemy presence. It iterates over the knowledge map in one pass finding the max likelihood and returns the associated way point. Figure 4.1 shows a typical flight pattern using this behavior control.

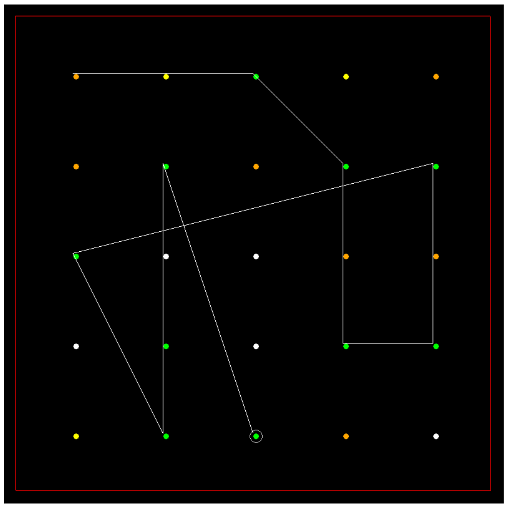


Figure 4.1: Route of simulated UAS (current location is the circle) searching a 5 x 5 node network using a *global greedy* behavior algorithm after ten glimpses. Notice how some legs span the width of the search box. Initial enemy likelihood was randomly assigned. Red represents belief of high probability of enemy presence, with yellow, white and green representing decreased belief of enemy presence. This picture was created by code titled virtual-CBTXXI_v2.4.py written in Python scripting language using the Tkinter package.

- `myopicGreedyModule.py` -guides the UAS to fly to the location that is a single arc length away with the greatest likelihood of enemy presence. Checks only the way points that are one arc length away, in this case these are identified by combination of looking +/- 1 away from the current way point, which is corresponds to keys in the dictionary. This would likely be done by geometry or sets in another application. Figure 4.2 shows a typical flight pattern using this behavior control.

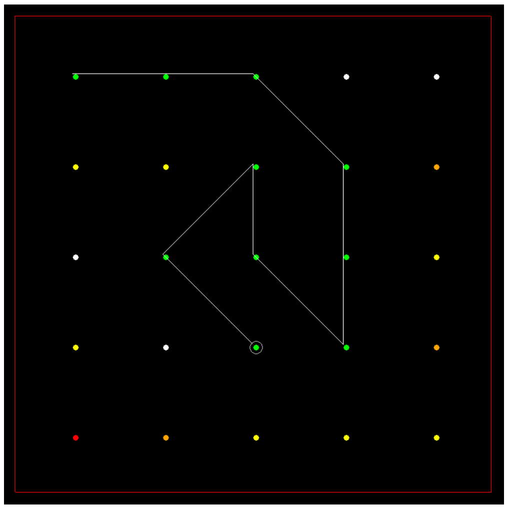


Figure 4.2: Route of simulated UAS (current location is the circle) searching a 5 x 5 node network using a *myopic greedy* behavior algorithm after ten glimpses. Only adjacent locations are considered when Initial enemy likelihood was randomly assigned. Red represents belief of high probability of enemy presence, with yellow, white and green representing decreased belief of enemy presence. This picture was created by code titled virtualCBTXXI_v2.4.py written in Python scripting language using the Tkinter package.

4. **Execute behavior.** In this Python only example, the move is simply an assumption the UAS arrives to the new location. This is confirmed with a print statement in the log and a circle drawn on the graphic at the new location.

5. **Sense the environment.** In this Python only example, there is no enemy target to find. However, there would be a return of 'found' or 'not found'. If 'found', the behavior should change to track this detected target or continue to search for more. This code contains only the case for 'not found'.

6. **Update knowledge map.** This demonstration uses a Bayesian update method. The method `updateBelief()` accomplishes this and is derived from code written by LTC Francisco Baez . This can be turned off with the Boolean variable `updateBelief`. It transfers the belief of the enemy in the cleared location and distributes it across the other locations.

   The idea of diffusion is another option used in updating the knowledge map. Diffusion considers the effects of a moving target. After a glimpse in one location, the target could move from where it was hiding into the location previously searched. The effect of this is similar to leveling of water in a cup. Essentially, each neighboring node in the network gives a percentage of its value to all its neighbors. In this code, it is done by reducing each location's level of belief by a percentage (say 10%) then adds to it that same percentage of each of its neighbors, divided by the number of neighbors. The method `diffuse()` accomplished this. This can be turned off with the Boolean variable `Diffusion`.

7. **Repeat steps 2 through 6.**

4.2. Application in COMBATXXI

The application of this capability is accomplished in COMBATXXI through BSL calls to supporting Python scripts. The TRAC-WSMR team contributed greatly to this integration and their work is preserved on the TRAC Knowledge management site under trac/Projects/643/. The effort to integrate the demonstration Python into a COMBATXXI scenario progressing through four phases.

4.2.1. Phase 1, Motivate Route Planning.

The first phase consisted of implementing non-dynamic route planning for a single decision making agent. The knowledge map is instantiated the same as the Python demonstration. The locations of knowledge map nodes are oriented in a 5 by 5 grid centered on the center of a generic scenario box. The likelihood of enemy presence at each node is initialized with randomly generated values using `fillRandom()`. The route for the agent is essentially determined at the beginning of the scenario. Without any mechanism to update $\lambda_{(n)}$ (likelihood), the route is set at the beginning of the scenario visiting $\gamma_{(n)}$ (locations) in descending order of $\lambda_{(n)}$. A UH-60 (chosen for its hover capability) called `DEFAULT_UH-60A_31` serves as our agent. The agent moves to the most likely location on the list, conducts a stationary search, then moves to the next location repeatedly until stopping criteria is met. Eleven threat entities (Red T-72 entities) are positioned in the scenario to observe search performance.

4.2.2. Phase 2, Updating the Knowledge Map.

For phase 2, the agent moves to a location, searches, *updates the knowledge map*, selects a next location, then moves to the next location. The route for the agent is not determined at the beginning of the scenario. The route is created one leg at a time as determined by the behavior `greedyModule.py` before the next move. This process continues until the agent has reached a max distance traveled as defined in the Python script. The original `flyroute()` Python method is separated into 3 methods in order to allow the interaction between model and script: `initialize_search()` , `continue_search()`, and `finish_search()`.

4.2.3. Phase 3, Multiple Agents with Transferred Knowledge Map.

For phase 3, multiple agents conduct knowledge map informed searches. The first agent conducts its search as in phase 2 and reaches a search stopping criteria, then a second agent, named `DEFAULT_UH-60A_77`, receives a copy of the knowledge map to conduct its search. The copied knowledge map dictates the search locations and initial likelihood of the second agents are the same as those of the first agent at the conclusion of its search. This is an example of multiple agents using a single behavior with separate knowledge maps. This procedure is best described in the following flow chart:

1. `KickOff` (APPENDIX I.1) BSL behavior triggered on `AllMyFMsHaveBeenInitialized`.
2. Executes `main.py`
   - Initializes knowledge map labeled `tmap`.
   - Calls `initialize_search()`
     - resets `distanceTot` and `glimpses` variables.
   - Calls `continue_search()` (APPENDIX I.2)
     - `continue_search()` replaces the while loop construct from the original python script.
     - Calls `greedyModule.py` to select the next way point to search.
     - Calls `cxxiintegration.cxxi_send_uav_to()` Sends agent to selected way point, hovers and searches for 300 seconds.
     - Calls `NewThreatDetected` (APPENDIX I.3) if target is detected.
     - Updates `tmap` from search results.
   - Call `finish_search()` if either `maxGlimpses` or `maxDistance` stopping criteria are met, ending the search for this agent.
3. Call `finish_search()`
   - The original `tmap` is copied and stored in a `LinkedHashMap` . The `tmap` itself can not be passed through the generic event.
   - This `LinkedHashMap` is passed by a `GenericEvent` and used to create a new knowledge map for the next agent (`DEFAULT_UH-60A_77`) to take over the search.
4. `DEFAULT_UH-60A_77` receives a `GenericEvent` generated by agent `DEFAULT_UH-60A_31`.
5. Values for `tmap` are passed in a `GenericEvent` in a `LinkedHashMap`.
6. Executes `main.py`
   - `main.py` is called to initialize the namespace making the necessary Python methods available.
   - Call `handoff_search()` (APPENDIX I.4) to rebuild another `tmap` using values from `LinkedHashMap`.

- Call `initialize_search()` and `continue_search()` to continue search pattern for second agent.
- If either `maxGlimpses` or `maxDistance` stopping criteria are met, the search is complete.

4.2.4. Phase 4, Multiple Agents, Single Knowledge Map.

For phase 4, multiple agents use the same knowledge map to inform their search. The knowledge map is instantiated in a class that allows access from every agent in the scenario. This removes the necessary procedure of copying `tmap` to a `LinkedHashMap` for the next agent to take over the search. This single universal knowledge map changes the procedure of passing knowledge between the agents. Even though the two agents can now modify the same knowledge map, we do not observe a change of behavior due to the sequential nature their searches in this scenario. This is not true when we apply this technique to an existing scenario, discussed later.

4.3. Using Knowledge Maps in Existing Scenarios

With success of implementing the knowledge map concept into a generic COMBATXXI environment, we investigate implementing them into an existing scenario. The Deployed Force Protection scenario will serve as the existing scenario. We discuss applying the techniques developed in phase 3 and 4 separately.

4.3.1. Phase 3 Techniques in an Existing Scenario

TRAC-MTRY implemented knowledge map informed UAS search behavior in an existing scenario using phase 3 techniques. The observed behavior proves reasonable. Two hours were required to implement a behavior that simulates UAS search to the scenario by two individuals possessing minimum experience with COMBATXXI. This time included adding the agent entities, the BSL code and modifying the Python code to center `tmap` on the combat outpost. As this technique is refined, less time is likely necessary.

Analysis of this scenario would not be complete if it supported a study. There is a deliberate process of verification and validation necessary to ensure the scenario is proper. Verification would check good general programming practice, and the intermediate trace and final outputs are produced properly. Validation would ensure the data (including behavior rules) reflect the effects of real world and the outputs are within statistical expected or feasible results [7] [5] [6] [8]. The log file located in APPENDIX J assists verifying the behavior.

We show a scenario developer can create a single behavior module for use in any scenario. Once created, this behavior can quickly be added to a scenario with minimum concern of invalid behavior or laborious re-scripting. The UAS decision making agent will use the knowledge map to assess the state in the new scenario and apply the behavior control to execute actions appropriate to the new scenario.

4.3.2. Phase 4 Techniques in an Existing Scenario

TRAC-MTRY implemented knowledge map informed UAS search behavior in an existing scenario using phase 4 techniques. The observed behavior was not as expected. Instead of two, separate sequential searches, we observed a coordinated search. The first agent initiated its search as expected. However, the second agent initiated its search soon after the first. Since the single knowledge map informed their behavior and both modified the knowledge map according to sensing effects they were likely simulating a coordinated search with perfectly shared information. The log file in APPENDIX J further describes this scenario. It is suspected a stray `GenericEvent` from the scenario initiated the search for the second agent. The TRAC-MTRY team did not investigate further. There is value in understanding that attention should be given to mitigating this risk before using this technique.

This page intentionally left blank.

5. RESULTS AND CONCLUSIONS

This project demonstrates a proof of concept use of graphs to represent knowledge enabling dynamic behaviors within COMBATXXI. This chapter discusses summary results, conclusions and finally future work.

5.1. Results

The following are our methodological steps and summary results.

1. **Defined knowledge maps and possible applications.** Military simulations supporting analysis should allow a means to gain insight and assist in answering study questions. Supporting this effort, knowledge maps promise to assist in rapid scenario development. Assembling excepted behaviors and knowledge maps into scenarios reduces limited development resources. It is feasible behaviors and the way they are connected and informed through knowledge maps can carry over between multiple agents in a scenario, across multiple scenarios and even multiple projects. This approach has potential to focus the scenario developers time on the specific needs of answering the research questions of the project, and not on the monotonous explicit scripting of agent behavior and control.

2. **Developed Python code knowledge map and behavior.** A simple set of Python scripts developed specifically for this project simulate UAS behavior with situational awareness.

3. **Created a generic COMBATXXI scenario that used the Python code.** This approach is particularly useful when creating many similar agents in the same scenario. Each agent would receive its own knowledge map representing its particular understanding of the simulated environment. This application is demonstrated in Figure 2.5. A single behavior control module would access the knowledge map associated to an agent and determine its next location for each agent separately.

4. **Integrated the Python and BSL methods into an existing COMBATXXI scenario.** The technique was applied to an existing scenario. Knowledge map informed decision making was relatively quickly integrated into the scenario that resulted in agent behavior appropriately applying its particular behavior within its authorized scope of information. We also show there are several secondary benefits to this methodology including representing collective awareness. Additional capability is to allow the sensed information processed from many entities to inform the decisions. This convergence of information allows for a representation of a group based knowledge of the environment and individual actions that reflect awareness of this greater knowledge of the group. This technical report documents the implementation of using knowledge to inform dynamic behaviors within COMBATXXI.

5.2. Conclusion

We demonstrate it is possible to implement non-prescriptive dynamic behavior through use of a knowledge map without modification to the COMBATXXI base code. There is great potential benefit to rapid scenario creation. We also show there are several secondary benefits to this methodology, including representing collective awareness.

5.3. Future Work

The use of knowledge maps will depend on future work addressing the following issues:

- Development and adoption of standards and procedures necessary for efficient, quick turnaround scenario building capability to increase agility and responsiveness of COMBATXXI.
- Methods to modularized knowledge maps and behaviors to represent common requirements for studies.
- Identify how use of knowledge map informed behavior impacts runtime.
- Determine a means to assess an appropriate use of knowledge maps according to echelon with attention to the number of knowledge maps and behavior modules to support a scenario.
- Application of state-space partitioning techniques, route planning and complex knowledge representation [4] [14].
- Application of artificial intelligence algorithms, such as Monte-Carlo Tree Search, for greater insight to analyzing the effects of changing parameters of a scenario on aspects of mission command [10].

APPENDIX A. virtualCBTXXI.py Script

```
#    Threat tmap Demonstration
#          In support of TRAC Project Code 642
#    author: Peter Nesbitt      TRAC-MTRY
#    peter.nesbitt@us.army.mil
#
#
# This program builds a dictionary representing a threat tmap,
# selects cells to move a sensor to (initially greedy algorithm),
# reduces the probability, then moves to the next cell.
#
#

#--------------------------------------------#
#             IMPORTS                        #
#--------------------------------------------#
# packages
from random import *
from Tkinter import *
import time
from math import *


# modules for behavior to place into CBTXXI
import greedyModule
# import waveFrontModule
import myopicGreedyModule
import myopicLeastModule
#--------------------------------------------#
#             GLOBAL CONSTANTS               #
#--------------------------------------------#

# The area to be searched is grid[1] by grid[0]
grid= 30, 30 # grid[0] * grid[1] nodes in rectangular shaped graph
window= 800, 800 # size of window for graphics
platLoc= 0,0 # initial location of UAV

# sensor capability
glimpseEffect= 1 # glimpseEffect is the placeholder for p infinity as calculated by Acquire
maxGlimpses= 4*grid[0]*grid[1]  # stopping criteria for number of glimpses allowed
maxDistance= 2*grid[0]*grid[1]  # stopping criteria for distance traveled

# Data structures, representing a network of locations
tmap= {}  # initialize empty probability dictionary
cmap= {}  # initialize empty color dictionary


# Setting for initial probability distribution for beleif
initialBelief= 3
# 1 uses fillUniform(): # Call to fill all cells with same uniform probability (Random Walk)
# 2 uses fillRandom(): # Informed initial tmap distribution (random gen per node) with
    belief one
# 3 uses fillCenter(): # normal distribution centered on network

# Settings for behavior in selecting the next WP to move the UAV to
behavior= 3
# 1 is greedy       # always fly to the greatest probability
# 2 is wavefront    # not ready
# 3 is myopicGreedy # greedy with very short sight, must move to another node per glimpse
# 4 is myopicTrapping # least greedy with very short sight, must move to another node per
    glimpse
# 5 is zamboni      # not ready
# 6 is lawnmower    # not ready

updateBelief= False # Bayesian update
```

```
Diffusion= False # Do % of probabilities 'diffuse' to adjacent cells?
difRate= .05     # rate of diffusion between adjacent cells, rate of loss in value of
    information
Norm= False     # re-Normalize after every glimpse?


## Settings for program output to user through text
printOn= True

 # Graphical settings for output to user through pictures
visual= True
if visual:
  pause= .5 # time in seconds per glimpse for visualization of search
  root = Tk()
  iconSize= 10
  pEx= 1.0,0.0 # set the low to 1 and the high to 0
  tolerance= 0
  extCol,highCol,midCol,lowCol,noneCol= 'red','orange','yellow','white','green'
  extProb,highProb,midProb,lowProb,= 0,0,0,0
  colorBins= extProb,highProb,midProb,lowProb
  fieldX= .9*window[0]
  fieldY= .9*window[1]
  unitX= float(fieldX)/float(grid[0])
  unitY= float(fieldY)/float(grid[1])
  baseX= .05*window[0]+ .5*unitX
  baseY= .05*window[1]+ .5*unitY


#--------------------------------------------#
#          FUNCTIONS AND CLASSES             #
#--------------------------------------------#

### ------  Graphical Functions ------  ###
def buildBbox(x,y):  # build canvas coordinates for ovals and rectangles
  bbox= (x*unitX+baseX,
         y*unitY+baseY,
         x*unitX+baseX+iconSize,
         y*unitY+baseY+iconSize)
  return bbox

def buildLine(WP1,WP2): # build canvas coordinates for lines between nodes
  bbox= (WP1[0]*unitX+baseX,
         WP1[1]*unitY+baseY,
         WP2[0]*unitX+baseX,
         WP2[1]*unitY+baseY)
  return bbox

def setColors(WP,pEx):  # identify the extreme probabilities
  if tmap[WP]<pEx[0]: # is the new prob lower than the low extreme?
    pEx= tmap[WP],pEx[1]
    span=pEx[1]-tmap[WP]
    if printOn:
      print 'l prob range changed to ', pEx
  if tmap[WP]>pEx[1]: # is the new prob higher than the high extreme?
    pEx= pEx[0],tmap[WP]
    span=tmap[WP]-pEx[0]
    if printOn:
      print 'u prob range changed to ', pEx
  else:
    span= pEx[1]-pEx[0]  # if no change, keep span as
    # set span to the range between high and low
  extProb,highProb,midProb,lowProb= .75*span+pEx[0],.5*span+pEx[0],.25*span+pEx[0],pEx[0]
  colorBins= extProb,highProb,midProb,lowProb
  if printOn:
      print 'setColors', WP, pEx
  return colorBins, pEx

def showNode(WP,colorBins):  # initial instantiation and show for nodes
```

```
  prob= tmap[WP]
  bbox=buildBbox(WP[0],WP[1])
  extProb,highProb,midProb,lowProb= colorBins
  if prob>= extProb:
    color= extCol
  elif prob>=highProb:
    color= highCol
  elif prob>=midProb:
    color= midCol
  elif prob>= lowProb:
    color= lowCol
  else :
    color= noneCol
  cmap[WP]= canv.create_oval(bbox, fill=color)
  if printOn:
    print 'showNode', WP,' ', "%0.3f" % (tmap[WP]), color, cmap[WP]
  item= cmap[WP]
  # time.sleep(pause/10)
  root.update()

def changeNodeColor(WP,colorBins): # subsequent change of color for nodes
  extProb,highProb,midProb,lowProb= colorBins
  prob= tmap[WP]
  item= cmap[WP]
  if prob>= extProb:
    color= extCol
  elif prob>=highProb:
    color= highCol
  elif prob>=midProb:
    color= midCol
  elif prob>= lowProb:
    color= lowCol
  else :
    color= noneCol
  canv.itemconfig(item, fill=color)
  root.update()
  if printOn:
    print 'updated',WP,' ', "%0.3f" % (tmap[WP]), color, item

def buildUAV(WP):  # initial UAV icon build
  x,y= WP[0],WP[1]
  if printOn:
    print 'buildUAV', WP
  uavLoc= (x*unitX+baseX-5,
       y*unitY+baseY-5,
       x*unitX+baseX+iconSize+5,
       y*unitY+baseY+iconSize+5)
  canv.create_oval(uavLoc, outline='gray',tags='uav')
  root.update()

def moveUAV(WP):  # subsequent UAV move to node
  canv.delete('uav')
  x,y= WP[0],WP[1]
  if printOn:
    print 'moveUAV', WP
  uavLoc= (x*unitX+baseX-5,
       y*unitY+baseY-5,
       x*unitX+baseX+iconSize+5,
       y*unitY+baseY+iconSize+5)
  canv.create_oval(uavLoc, outline='gray',tags='uav')
  root.update()

def buildCanvas(): # Instantiation and settings for canvas
  Label(text="Threat Map Demonstration").pack()
  searchareaX= float(1.2*window[0])
  searchareaY= float(1.2*window[1])
  canv = Canvas(root, height=window[1], width=window[0], bg="black")
  canv.pack()
  Button(text="Dismiss", command=root.quit).pack(side=LEFT)
  tlc= .025*window[0], .025*window[1]
  trc= .975*window[0], .025*window[1]
  blc= .025*window[0], .975*window[1]
  brc= .975*window[0], .975*window[1]
  canv.create_line(tlc, trc, fill="red", width=1)
  canv.create_line(tlc, blc, fill="red", width=1)
  canv.create_line(brc, trc, fill="red", width=1)
  canv.create_line(brc, blc, fill="red", width=1)
  root.update()
  return canv

### ------  Printing Functions ------  ###

def reporttmap():  # print the keys and values of all nodes in tmap
  for key, value in tmap.iteritems():
    print key, "%0.3f" % (value)

### ------  Search related Functions ------  ###

def fillUniform(pEx): # Call to fill all cells with uniform probability
  print 'fillUniform'
  for y in range(grid[1]):
    for x in range(grid[0]):
        nWP= x,y
        tmap[nWP]= 1/(float(grid[1])*float(grid[0]))
        if visual:
          if tmap[nWP]>highProb or tmap[nWP]<lowProb:
            colorBins, pEx= setColors(nWP,pEx)
  return colorBins, pEx

def fillRandom(pEx): # Informed initial tmap distribution (random gen) with belief one
  print 'fillRandom'
  for y in range(grid[1]):
    for x in range(grid[0]):
        nWP= x,y
        tmap[nWP] = uniform(0,1)
        if visual:
          if tmap[nWP]>highProb or tmap[nWP]<lowProb:
            colorBins, pEx= setColors(nWP,pEx)
  return colorBins, pEx

def normpdf(x, mu, sigma):
  u = (x-mu)/abs(sigma)
  y = (1/(sqrt(2*pi)*abs(sigma)))*exp(-u*u/2)
  return y

def fillNormalCenter(pEx):
  print 'fillNormalCenter'
  xmu, ymu= .5*grid[0],.5*grid[1]
  print xmu
  xsigma, ysigma= .1*grid[0], .1*grid[1]
  for y in range(grid[1]):
    for x in range(grid[0]):
      print x,y, normpdf(x+.5,xmu,xsigma)+normpdf(y+.5,ymu,ysigma)
      nWP= x,y

      tmap[nWP]= normpdf(x+.5,xmu,xsigma)+normpdf(y+.5,ymu,ysigma)
      colorBins, pEx= setColors(nWP,pEx)
      # print nWP, tmap[nWP]
  return colorBins, pEx

def normProb() :  # Normalize probability for unsearched cells
  sum=0
  for y in range(grid[1]):
    for x in range(grid[0]):
      WP= x,y
      sum+= tmap[WP]
  for y in range(grid[1]):
    for x in range(grid[0]):
      WP= x,y
      if tmap[WP]!= 0:
        tmap[WP] = tmap[WP]/sum

def diffuse(WP,colorBins): # allow a fraction of beleif to move between adjacent nodes
  if printOn:
    print 'start diffuse'
    reporttmap()
  tmap2=tmap
  for key, value in tmap.iteritems():
    for i in range(3):
      for j in range(3):
        dWP=key[0]+(i-1),key[1]+(j-1)
        if dWP== key: # reduce itself by 1-difRate
          tmap[key]= (1-difRate)*tmap2[key]
        elif dWP in tmap:
          tmap[key]= value+ (difRate*tmap2[dWP])/8.0
    changeNodeColor(key,colorBins)
  if printOn:
    print 'diffuse complete'
    reporttmap()

def updateBelief(WP, glimpseEffect, tmap):
  # Derived from Francisco Baez Toledo, MOVES, NPS
  # Derived from Bayesian update
  # xLoc and yLoc represent the current cell inspected
  # glimpseEffect is the placeholder for p infinity as calculated by Acquire
  if printOn:
    print 'Bayesian belief update'
  ntmap= tmap
  s = 0
  #calculate non-normalized posterior probability
  for key, value in tmap.iteritems():
      if key == WP:
        hit = 0
      else:
        hit = 1
      ntmap[key]= value*hit + value*(1 - glimpseEffect) * (1 - hit)
      s += ntmap[key]
  #calculate normalized posterior probability
  for key, value in ntmap.iteritems():
    ntmap[key]= value / s
    changeNodeColor(key,colorBins)
  return ntmap

def distanceLeg(platLoc,WP): # calculates straight line distance between nodes
  dist= (((platLoc[0]-WP[0])**2)+((platLoc[1]-WP[1])**2))**.5
  return dist

def findWeightedMax():  # finds max in tmap dictionary
  # future editions may weigh distance for gain
  # WP= 0,0
  fWP= 0,0
  firstMax= 0
  # secondMax= 0
  for y in range(grid[1]):
    for x in range(grid[0]):
      testWP= x,y
      if tmap[testWP] > firstMax:
        firstMax= tmap[testWP]
        fWP= x,y
  return fWP

def flyRoute(platLoc, tmap):
  ## --- Initial Conditions to execute search
  search= True  # Start the search
```

```
distanceTot= 0
glimpses= 0

if behavior==1: # always fly to the greatest probability
  print 'global greedy method'# Find next highest probability
if behavior==2: # not ready
  print 'wavefront method not ready, using global greedy' # Find next highest probability
if behavior==3: # greedy with very short sight, must move to another node per glimpse
  print 'myopic greedy method'
if behavior==4: # least greedy with very short sight, must move to another node per
    glimpse
  print 'myopic trapping method'
if behavior==5: # shifting concentric circling for consistant wide turns
  print 'zamboni method' # not ready
if behavior==6: # start in one corner, up and down rows across network
  print 'lawnmower method' # not ready

# Identify first destination
WP= platLoc
if visual:
  buildUAV(platLoc)
  for key, value in tmap.iteritems():
    showNode(key,colorBins)

## --- Logic Loop controlling continued search
while search: # While search== True, continue to search
  if behavior==1:
    WP= greedyModule.findMaxWP(tmap) # Find next highest probability
  if behavior==3:
    WP= myopicGreedyModule.myopicGreedy(tmap,platLoc)
  if behavior==4:
    WP= myopicLeastModule.myopicLeast(tmap,platLoc,tolerance)

  # UAV will stop if all it can see is less than acceptable tolerance
  if tmap[WP]< tolerance :  # Completion criteria
    print tmap[WP],'<',tolerance,' Tolerance met in all cells.'
    search= False # search complete

  if printOn: # report action
      print 'Send UAV to', WP, ' with ', "%0.3f" % (tmap[WP])
  # update distance traveled
  distanceTot+= distanceLeg(platLoc,WP)
  glimpses+= 1
  if printOn:
    print 'dist= ',distanceTot,' glimpse= ',glimpses
  # display cleared status on canvas
  if visual:
    moveUAV(WP)
    bboxL= buildLine(platLoc,WP)
    canv.create_line(bboxL, fill="white", width=1)
    root.update()
    time.sleep(pause) # slow the search for visualization
  # glimpse at location WP reduces its probability
  if updateBelief== True:
    tmap = updateBelief(WP, glimpseEffect, tmap)
  else:
    # probabilities diffuse between locations during movement since last assessment
    if printOn:
      print 'No Bayesian update'
    if Diffusion:
      diffuse(WP,colorBins)
    tmap[WP]= (1-glimpseEffect)*tmap[WP]
    # normalize the tmap after glimpse
    if Norm:
      normProb()

  platLoc= WP    # UAV now at goal
  if printOn: # report action
```

```
      print 'UAV clears ', WP,' now ', "%0.3f" % (tmap[WP])

    # Update all colors if necessaruy
    if visual:
      if Diffusion or updateBelief==True:
        for key, value in tmap.iteritems():
          changeNodeColor(key,colorBins)
      else:
        changeNodeColor(WP,colorBins)


    if glimpses>=maxGlimpses :  # Completion criteria
      print "Max glimpses reached."
      search= False # search complete
    if distanceTot>= maxDistance: # Completion criteria
      print "Max distance reached."
      search= False # search complete
  print 'Return UAV to base.'
  return distanceTot, glimpses


#--------------------------------------------#
#            M A I N   B L O C K             #
#--------------------------------------------#

if visual:
  canv= buildCanvas() # create tk graphic canvas

# Fill tmap with initial belief of target location
if initialBelief== 1:
  colorBins, pEx= fillUniform(pEx) # Call to fill all cells with uniform probability
if initialBelief== 2:
  colorBins, pEx= fillRandom(pEx) # Informed initial tmap distribution (random gen per node)
       with belief one
if initialBelief== 3:
  colorBins, pEx= fillNormalCenter(pEx) # normal distribution centered on network
  # normProb()

if Norm:
  normProb()

if printOn:
  print 'Initial threat map probabilities:'
  reporttmap() # print tmap data to screen

distanceTot, glimpses= flyRoute(platLoc, tmap) # execute search algorithm

if printOn:
  print 'Final threat map probabilities:'
  reporttmap() # print tmap data to screen

print 'Complete, total distance traveled: ', "%0.3f" % (distanceTot), ' glimpses: ',
    glimpses

if visual:
  root.mainloop()
```

APPENDIX B. greedyModule.py Script

```
#    Knowledge Representation Demonstration in Python
#    Greedy Behavior Module
#    In support of TRAC Project Code 642
#    author: Peter Nesbitt     TRAC-MTRY
#    peter.nesbitt@us.army.mil
#
#
# This program represents a strictly greedy algorithm
# to move a sensor to the next location to search.

def findMaxWP(tmap) :
  maxProb= 0
  # print "greedyModule"
  for key, value in tmap.iteritems():
      if value > maxProb:
        maxProb= value
        WP= key
  return WP
```

APPENDIX C. myopicGreedyModule.py Script

```
#    Knowledge Representation Demonstration in Python
#    Myopic Greedy Behavior Module
#    In support of TRAC Project Code 642
#    author: Peter Nesbitt     TRAC-MTRY
#    peter.nesbitt@us.army.mil
#
#
# This program represents a myopic greedy agorithm
# to move a sensor to the next location to search.

def myopicGreedy(tmap,platLoc) :
  print 'start myopicGreedy'
  WP= platLoc
  maxProb= tmap[platLoc]
  xlook= (0,1,1,1,0,-1,-1,-1)
  ylook= (-1,-1,0,1,1,1,0,-1)
  for i in range(8):
    target= (platLoc[0]+xlook[i],platLoc[1]+ylook[i])
    if (target) in tmap:
      if tmap[target] > maxProb:
        WP= target
        maxProb= tmap[target]
  return WP
```

APPENDIX D. main.py Script

```
#    Threat tmap Demonstration
#           In support of TRAC Project Code 642
#    author: Peter Nesbitt     TRAC-MTRY
#    peter.nesbitt@us.army.mil
#
#
# This program builds a dictionary representing a threat tmap,
# selects cells to move a sensor to (initially greedy algorithm),
# reduces the probability, then moves to the next cell.
#
#

#--------------------------------------------#
#             IMPORTS                        #
#--------------------------------------------#
# packages
from random import *
#from Tkinter import *
import time
from math import *

# modules for behavior to place into CBTXXI
import greedyModule
# import waveFrontModule
import myopicGreedyModule
import cxxiintegration
#import Debugger
#import myopicLeastModule
#--------------------------------------------#
#             GLOBAL CONSTANTS               #
#--------------------------------------------#

orders2=orders
obs2=obs
state2=state
# The area to be searched is grid[1] by grid[0]
grid= 5, 5 # grid[0] * grid[1] nodes in rectangular shaped graph
window= 800, 800 # size of window for graphics
platLoc= 0,0 # initial location of UAV

# sensor capability
glimpseEffect= 1 # glimpseEffect is the placeholder for p infinity as calculated by Acquire
maxGlimpses= 4*grid[0]*grid[1]  # stopping criteria for number of glimpses allowed
#maxDistance= 2*grid[0]*grid[1]  # stopping criteria for distance traveled
maxDistance=100000
# Data structures, representing a network of locations
tmap= {}  # initialize empty probability dictionary
cmap= {}  # initialize empty color dictionary


# Setting for initial probability distribution for beleif
initialBelief= 3
# 1 uses fillUniform(): # Call to fill all cells with same uniform probability (Random Walk)
# 2 uses fillRandom(): # Informed initial tmap distribution (random gen per node) with
    belief one
# 3 uses fillCenter(): # normal distribution centered on network

# Settings for behavior in selecting the next WP to move the UAV to
behavior= 1
# 1 is greedy       # always fly to the greatest probability
# 2 is wavefront    # not ready
# 3 is myopicGreedy # greedy with very short sight, must move to another node per glimpse
# 4 is myopicTrapping # least greedy with very short sight, must move to another node per
    glimpse
# 5 is zamboni      # not ready
# 6 is lawnmower    # not ready

updateBelief= True # Bayesian update

Diffusion= True # Do % of probabilities 'diffuse' to adjacent cells?
difRate= .05     # rate of diffusion between adjacent cells, rate of loss in value of
    information
Norm= False     # re-Normalize after every glimpse?


## Settings for program output to user through text
printOn= True

 # Graphical settings for output to user through pictures
visual= False
if visual:
  pause= .5 # time in seconds per glimpse for visualization of search
  root = Tk()
  iconSize= 10


  extCol,highCol,midCol,lowCol,noneCol= 'red','orange','yellow','white','green'
  extProb,highProb,midProb,lowProb,= 0,0,0,0
  colorBins= extProb,highProb,midProb,lowProb
  fieldX= .9*window[0]
  fieldY= .9*window[1]
  unitX= float(fieldX)/float(grid[0])
  unitY= float(fieldY)/float(grid[1])
  baseX= .05*window[0]+ .5*unitX
  baseY= .05*window[1]+ .5*unitY

pEx= 1.0,0.0 # set the low to 1 and the high to 0
tolerance= 0
#---------------------------------------------#
#          FUNCTIONS AND CLASSES              #
#---------------------------------------------#

### ------  Graphical Functions ------  ###
def buildBbox(x,y):  # build canvas coordinates for ovals and rectangles
  bbox= (x*unitX+baseX,
         y*unitY+baseY,
         x*unitX+baseX+iconSize,
         y*unitY+baseY+iconSize)
  return bbox

def buildLine(WP1,WP2): # build canvas coordinates for lines between nodes
  bbox= (WP1[0]*unitX+baseX,
         WP1[1]*unitY+baseY,
         WP2[0]*unitX+baseX,
         WP2[1]*unitY+baseY)
  return bbox

def setColors(WP,pEx):  # identify the extreme probabilities
  if tmap[WP]<pEx[0]: # is the new prob lower than the low extreme?
    pEx= tmap[WP],pEx[1]
    span=pEx[1]-tmap[WP]
    if printOn:
      print 'l prob range changed to ', pEx
  if tmap[WP]>pEx[1]: # is the new prob higher than the high extreme?
    pEx= pEx[0],tmap[WP]
    span=tmap[WP]-pEx[0]
    if printOn:
      print 'u prob range changed to ', pEx
  else:
    span= pEx[1]-pEx[0]   # if no change, keep span as
    # set span to the range between high and low
  extProb,highProb,midProb,lowProb= .75*span+pEx[0],.5*span+pEx[0],.25*span+pEx[0],pEx[0]
  colorBins= extProb,highProb,midProb,lowProb
  if printOn:
      print 'setColors', WP, pEx
  return colorBins, pEx

def showNode(WP,colorBins):  # initial instantiation and show for nodes
  prob= tmap[WP]
  bbox=buildBbox(WP[0],WP[1])
  extProb,highProb,midProb,lowProb= colorBins
  if prob>= extProb:
    color= extCol
  elif prob>=highProb:
    color= highCol
  elif prob>=midProb:
    color= midCol
  elif prob>= lowProb:
    color= lowCol
  else :
    color= noneCol
  cmap[WP]= canv.create_oval(bbox, fill=color)
  if printOn:
    print 'showNode', WP,' ', "%0.3f" % (tmap[WP]), color, cmap[WP]
  item= cmap[WP]
  # time.sleep(pause/10)
  root.update()

def changeNodeColor(WP,colorBins): # subsequent change of color for nodes
  extProb,highProb,midProb,lowProb= colorBins
  prob= tmap[WP]
  item= cmap[WP]
  if prob>= extProb:
    color= extCol
  elif prob>=highProb:
    color= highCol
  elif prob>=midProb:
    color= midCol
  elif prob>= lowProb:
    color= lowCol
  else :
    color= noneCol
  canv.itemconfig(item, fill=color)
  root.update()
  if printOn:
    print 'updated',WP,' ', "%0.3f" % (tmap[WP]), color, item

def buildUAV(WP):  # initial UAV icon build
  x,y= WP[0],WP[1]
  if printOn:
    print 'buildUAV', WP
  uavLoc= (x*unitX+baseX-5,
       y*unitY+baseY-5,
       x*unitX+baseX+iconSize+5,
       y*unitY+baseY+iconSize+5)
  canv.create_oval(uavLoc, outline='gray',tags='uav')
  root.update()

def moveUAV(WP):  # subsequent UAV move to node
  canv.delete('uav')
  x,y= WP[0],WP[1]
  if printOn:
    print 'moveUAV', WP
  uavLoc= (x*unitX+baseX-5,
       y*unitY+baseY-5,
       x*unitX+baseX+iconSize+5,
       y*unitY+baseY+iconSize+5)
  canv.create_oval(uavLoc, outline='gray',tags='uav')
  root.update()

def buildCanvas(): # Instantiation and settings for canvas
```

```
  Label(text="Threat Map Demonstration").pack()
  searchareaX= float(1.2*window[0])
  searchareaY= float(1.2*window[1])
  canv = Canvas(root, height=window[1], width=window[0], bg="black")
  canv.pack()
  Button(text="Dismiss", command=root.quit).pack(side=LEFT)
  tlc= .025*window[0], .025*window[1]
  trc= .975*window[0], .025*window[1]
  blc= .025*window[0], .975*window[1]
  brc= .975*window[0], .975*window[1]
  canv.create_line(tlc, trc, fill="red", width=1)
  canv.create_line(tlc, blc, fill="red", width=1)
  canv.create_line(brc, trc, fill="red", width=1)
  canv.create_line(brc, blc, fill="red", width=1)
  root.update()
  return canv

### ------  Printing Functions ------  ###

def reporttmap():  # print the keys and values of all nodes in tmap
  for key, value in tmap.iteritems():
    print key, "%0.3f" % (value)

### ------  Search related Functions ------  ###

def fillUniform(pEx): # Call to fill all cells with uniform probability
  print 'fillUniform'
  for y in range(grid[1]):
    for x in range(grid[0]):
        nWP= x,y
        tmap[nWP]= 1/(float(grid[1])*float(grid[0]))
        if visual:
          if tmap[nWP]>highProb or tmap[nWP]<lowProb:
            colorBins, pEx= setColors(nWP,pEx)
  return colorBins, pEx

def fillRandom(pEx): # Informed initial tmap distribution (random gen) with belief one
  print 'fillRandom'
  for y in range(grid[1]):
    for x in range(grid[0]):
        nWP= x,y
        tmap[nWP] = uniform(0,1)
        if visual:
          if tmap[nWP]>highProb or tmap[nWP]<lowProb:
            colorBins, pEx= setColors(nWP,pEx)
  return colorBins, pEx

def normpdf(x, mu, sigma):
  u = (x-mu)/abs(sigma)
  y = (1/(sqrt(2*pi)*abs(sigma)))*exp(-u*u/2)
  return y

def fillNormalCenter(pEx):
  print 'fillNormalCenter'
  xmu, ymu= .5*grid[0],.5*grid[1]
  print xmu
  xsigma, ysigma= .1*grid[0], .1*grid[1]
  for y in range(grid[1]):
    for x in range(grid[0]):
      print x,y, normpdf(x+.5,xmu,xsigma)+normpdf(y+.5,ymu,ysigma)
      nWP= x,y

      tmap[nWP]= normpdf(x+.5,xmu,xsigma)+normpdf(y+.5,ymu,ysigma)
      colorBins, pEx= setColors(nWP,pEx)
      # print nWP, tmap[nWP]
  return colorBins, pEx

def normProb() :  # Normalize probability for unsearched cells
```

```
  sum=0
  for y in range(grid[1]):
    for x in range(grid[0]):
      WP= x,y
      sum+= tmap[WP]
  for y in range(grid[1]):
    for x in range(grid[0]):
      WP= x,y
      if tmap[WP]!= 0:
        tmap[WP] = tmap[WP]/sum

def diffuse(WP,colorBins): # allow a fraction of beleif to move between adjacent nodes
  if printOn:
    print 'start diffuse'
    reporttmap()
  tmap2=tmap
  for key, value in tmap.iteritems():
    for i in range(3):
      for j in range(3):
        dWP=key[0]+(i-1),key[1]+(j-1)
        if dWP== key: # reduce itself by 1-difRate
          tmap[key]= (1-difRate)*tmap2[key]
        elif dWP in tmap:
          tmap[key]= value+ (difRate*tmap2[dWP])/8.0
    if visual:
            changeNodeColor(key,colorBins)
  if printOn:
    print 'diffuse complete'
    reporttmap()

def updateBelief(WP, glimpseEffect, tmap):
  # Derived from Francisco Baez Toledo, MOVES, NPS
  # Derived from Bayesian update
  # xLoc and yLoc represent the current cell inspected
  # glimpseEffect is the placeholder for p infinity as calculated by Acquire
  if printOn:
    print 'Bayesian belief update'
  ntmap= tmap
  s = 0
  #calculate non-normalized posterior probability
  for key, value in tmap.iteritems():
      if key == WP:
        hit = 0
      else:
        hit = 1
      ntmap[key]= value*hit + value*(1 - glimpseEffect) * (1 - hit)
      s += ntmap[key]
  #calculate normalized posterior probability
  for key, value in ntmap.iteritems():
    ntmap[key]= value / s
    #changeNodeColor(key,colorBins)
  return ntmap

def distanceLeg(platLoc,WP): # calculates straight line distance between nodes
  dist= (((platLoc[0]-WP[0])**2)+((platLoc[1]-WP[1])**2))**.5
  return dist

def findWeightedMax():  # finds max in tmap dictionary
  # future editions may weigh distance for gain
  # WP= 0,0
  fWP= 0,0
  firstMax= 0
  # secondMax= 0
  for y in range(grid[1]):
    for x in range(grid[0]):
      testWP= x,y
      if tmap[testWP] > firstMax:
        firstMax= tmap[testWP]
        fWP= x,y
  return fWP



#
    ####################################################################################################
# search now needs to be a module variable

def initialize_search():
  print 'initialize_search() called'
  ## --- Initial Conditions to execute search
  #search= True  # Start the search
  #distanceTot= 0
  #glimpses= 0
  if behavior==1: # always fly to the greatest probability
    print 'global greedy method'# Find next highest probability
  if behavior==2: # not ready
    print 'wavefront method not ready, using global greedy' # Find next highest probability
  if behavior==3: # greedy with very short sight, must move to another node per glimpse
    print 'myopic greedy method'
  if behavior==4: # least greedy with very short sight, must move to another node per
      glimpse
    print 'myopic trapping method'
  if behavior==5: # shifting concentric circling for consistant wide turns
    print 'zamboni method' # not ready
  if behavior==6: # start in one corner, up and down rows across network
    print 'lawnmower method' # not ready

distanceTot=0
glimpses=0

# used to pass updated tmap to entity taking over search
def initialize_search2(new_tmap):
  tmap=new_tmap
  initialize_search()

def continue_search():
  global search
  global platLoc
  global tmap
  global distanceTot
  global glimpses
  global orders
  global obs
  global state

  # Identify first destination
  WP=platLoc

  ## --- Logic Loop controlling continued search
  # not a loop anymore
  if search: # While search== True, continue to search
    if behavior==1:
      WP= greedyModule.findMaxWP(tmap) # Find next highest probability
    if behavior==3:
      WP= myopicGreedyModule.myopicGreedy(tmap,platLoc)
    if behavior==4:
      WP= myopicLeastModule.myopicLeast(tmap,platLoc,tolerance)

    # UAV will stop if all it can see is less than acceptable tolerance
    if tmap[WP]< tolerance :  # Completion criteria
      print tmap[WP],'<',tolerance,' Tolerance met in all cells.'
      search= False # search complete

    if printOn: # report action
        print 'Send UAV to', WP, ' with ', "%0.3f" % (tmap[WP])
    cxxiDistanceTraveled=cxxiintegration.cxxi_send_uav_to(orders,obs,state,WP)

    # update distance traveled
    #distanceTot+= distanceLeg(platLoc,WP)
    distanceTot+=cxxiDistanceTraveled
    glimpses+= 1
    if printOn:
      print 'dist= ',distanceTot,' glimpse= ',glimpses

    # glimpse at location WP reduces its probability
    if updateBelief== True:
      tmap = updateBelief(WP, glimpseEffect, tmap)
    else:
      # probabilities diffuse between locations during movement since last assessment
      if printOn:
        print 'No Bayesian update'
      if Diffusion:
        diffuse(WP,colorBins)
      tmap[WP]= (1-glimpseEffect)*tmap[WP]
      # normalize the tmap after glimpse
      if Norm:
        normProb()

    platLoc=WP    # UAV now at goal
    if printOn: # report action
      print 'UAV clears ', WP,' now ', "%0.3f" % (tmap[WP])

    if glimpses>=maxGlimpses :  # Completion criteria
      print "Max glimpses reached."
      search= False # search complete
    if distanceTot>= maxDistance: # Completion criteria
      print "Max distance reached."
      search= False # search complete

    if not search:
      print 'Return UAV to base.'
      finish_search()
  else:
    finish_search()
  #return distanceTot, glimpses

def finish_search():
  print 'Search Complete'

def new_threat_detected():
  global obs
  observedEntities=obs.getObsBySensorName('BINOC/7X')
  # glimpseEffect is the placeholder for p infinity as calculated by Acquire
  # if there are an observed entities then set the glimpseEffect variable
  # to the p-infinity value of the first observed entity
  # this is probably not correct but serves as a placeholder
  if observedEntities.size()>0:
    glimpseEffect=observedEntities[0].getPinf()
    print 'Threat detected'
  else:
    glimpseEffect=0
    print 'Threat not detected'

#
    ##############################################################################################################



# def flyRoute(platLoc, tmap):
  # ## --- Initial Conditions to execute search
  # search= True  # Start the search
  # distanceTot= 0
  # glimpses= 0

  # if behavior==1: # always fly to the greatest probability
    # print 'global greedy method'# Find next highest probability
  # if behavior==2: # not ready
    # print 'wavefront method not ready, using global greedy' # Find next highest
        probability
  # if behavior==3: # greedy with very short sight, must move to another node per glimpse
    # print 'myopic greedy method'
  # if behavior==4: # least greedy with very short sight, must move to another node per
      glimpse
    # print 'myopic trapping method'
  # if behavior==5: # shifting concentric circling for consistant wide turns
    # print 'zamboni method' # not ready
  # if behavior==6: # start in one corner, up and down rows across network
    # print 'lawnmower method' # not ready

  # # Identify first destination
  # WP= platLoc

  # ## --- Logic Loop controlling continued search
  # while search: # While search== True, continue to search
    # if behavior==1:
      # WP= greedyModule.findMaxWP(tmap) # Find next highest probability
    # if behavior==3:
      # WP= myopicGreedyModule.myopicGreedy(tmap,platLoc)
    # if behavior==4:
      # WP= myopicLeastModule.myopicLeast(tmap,platLoc,tolerance)

    # # UAV will stop if all it can see is less than acceptable tolerance
    # if tmap[WP]< tolerance :  # Completion criteria
      # print tmap[WP],'<',tolerance,' Tolerance met in all cells.'
      # search= False # search complete

    # if printOn: # report action
        # print 'Send UAV to', WP, ' with ', "%0.3f" % (tmap[WP])



# ##########################################################################
# # Code added
    # cxxiDistanceTraveled=cxxiintegration.cxxi_send_uav_to(orders,obs,state,WP)
    # print cxxiDistanceTraveled
# ##########################################################################


    # # update distance traveled
    # #distanceTot+= distanceLeg(platLoc,WP)
    # distanceTot+=cxxiDistanceTraveled
    # glimpses+= 1
    # if printOn:
      # print 'dist= ',distanceTot,' glimpse= ',glimpses

    # # glimpse at location WP reduces its probability
    # if updateBelief== True:
      # tmap = updateBelief(WP, glimpseEffect, tmap)
    # else:
      # # probabilities diffuse between locations during movement since last assessment
      # if printOn:
        # print 'No Bayesian update'
      # if Diffusion:
        # diffuse(WP,colorBins)
      # tmap[WP]= (1-glimpseEffect)*tmap[WP]
      # # normalize the tmap after glimpse
      # if Norm:
        # normProb()
    # platLoc= WP    # UAV now at goal
    # if printOn: # report action
      # print 'UAV clears ', WP,' now ', "%0.3f" % (tmap[WP])

    # if glimpses>=maxGlimpses :  # Completion criteria
      # print "Max glimpses reached."
      # search= False # search complete
    # if distanceTot>= maxDistance: # Completion criteria
      # print "Max distance reached."
      # search= False # search complete


  # print 'Return UAV to base.'
  # return distanceTot, glimpses










#--------------------------------------------#
#            M A I N   B L O C K             #
#--------------------------------------------#
if visual:
  canv= buildCanvas() # create tk graphic canvas

# Fill tmap with initial belief of target location
if initialBelief== 1:
  colorBins, pEx= fillUniform(pEx) # Call to fill all cells with uniform probability
if initialBelief== 2:
  colorBins, pEx= fillRandom(pEx) # Informed initial tmap distribution (random gen per node)
       with belief one
if initialBelief== 3:
  colorBins, pEx= fillNormalCenter(pEx) # normal distribution centered on network
  # normProb()

if Norm:
  normProb()

if printOn:
  print 'Initial threat map probabilities:'
  reporttmap() # print tmap data to screen

# initialize the flyRoute class with everything necessary

search=True
initialize_search()
continue_search()

#distanceTot, glimpses= flyRoute(platLoc, tmap) # execute search algorithm



# if printOn:
  # print 'Final threat map probabilities:'
  # reporttmap() # print tmap data to screen

# print 'Complete, total distance traveled: ', "%0.3f" % (distanceTot), ' glimpses: ',
    glimpses

# if visual:
  # root.mainloop()
```

APPENDIX E. cxxiintegration.py Script

```
from cxxi.model.behavior import OrderUtilities
from cxxi.model.behavior import CompoundOrder
from cxxi.model.physics.geometry import Location
from cxxi.support.types import OrderTriggerType
import orient_sensor
from java.util import Vector
from cxxi.model.behavior.primitiveorders import ObservePO
from cxxi.model.behavior.primitiveorders import StopObservePO
from cxxi.model.behavior.primitiveorders import AddRulePO

def cxxi_send_uav_to(orders,obs,state,WP):
  orient_sensor.straight_down(obs)
  print 'Calling cxxi_send_uav_to'
  current_location=state.getCurrentLocation()
  terrain_location=translate_grid_to_UTM(WP)
  fly_to_location(orders,terrain_location)
  hover(orders)
  return terrain_location.distanceTo(current_location)

def translate_grid_to_UTM(WP):
  easting_center=385986
  northing_center=3764269
  print WP[0],WP[1]
  easting=easting_center+(WP[0]*1000)-2000
  northing=northing_center-((WP[1]*1000)-2000)
  return Location.getInstanceFromUTMValues(easting,northing,1500)

def fly_to_location(orders,to_location):
  print 'UAS flying to easting',to_location
  prims = Vector()
  prims.add(OrderUtilities.createMovePrimitive(to_location,10))
  order = CompoundOrder("Move to Location", OrderTriggerType.TIME_ORDER, 0.0, prims)
  orders.unitAddOrder(order)

def hover(orders):
  prims = Vector()
  o = ObservePO()
  o.setSensor('BINOC/7X')
  o.setDir(0)
  o.setFOR(360)
  o.setFixed(0)
  so = StopObservePO()
  so.setSensor('BINOC/7X')

  # primitive to Hover
  prims.add(OrderUtilities.createHoverPrimitive())
  # primitive to start observing
  prims.add(o)
  # primitive to wait
  prims.add(OrderUtilities.createWaitPrimitive(300))
  # primitive to stop observing
  prims.add(so)
  # check if any threat were detected
  ar=AddRulePO()
  ar.setRuleId("NewThreatDetected")
  prims.add(ar)
  ar=AddRulePO()
  ar.setRuleId("ContinueSearch")
  prims.add(ar)
  order = CompoundOrder("Hovering", OrderTriggerType.TIME_ORDER, 0.0, prims)
  orders.unitAddOrder(order)
```

APPENDIX F. flyRoute.py Script

```
import greedyModule
# import waveFrontModule
import myopicGreedyModule
import cxxiintegration

def __init__(platLoc,tmap,orders,obs,state):
  platLoc=platLoc
  self.tmap=tmap
  self.WP=platLoc

def initialize_search(self,behavior):
  self.behavior=behavior
  self.search=True
  self.distanceTot=0
  self.glimpses=0

  if behavior==1: # always fly to the greatest probability
    print 'global greedy method'# Find next highest probability
  if behavior==2: # not ready
    print 'wavefront method not ready, using global greedy' # Find next highest probability
  if behavior==3: # greedy with very short sight, must move to another node per glimpse
    print 'myopic greedy method'
  if behavior==4: # least greedy with very short sight, must move to another node per
      glimpse
    print 'myopic trapping method'
  if behavior==5: # shifting concentric circling for consistant wide turns
    print 'zamboni method' # not ready
  if behavior==6: # start in one corner, up and down rows across network
    print 'lawnmower method' # not ready

  # Identify first destination
  self.WP=self.platLoc

def continue_search(self):
  if self.search: # if search== True, continue to search
    if self.behavior==1:
      WP= greedyModule.findMaxWP(tmap) # Find next highest probability
    if self.behavior==3:
      WP= myopicGreedyModule.myopicGreedy(tmap,platLoc)
    if self.behavior==4:
      WP= myopicLeastModule.myopicLeast(tmap,platLoc,tolerance)

    # UAV will stop if all it can see is less than acceptable tolerance
    if self.tmap[self.WP]< tolerance :  # Completion criteria
      print self.tmap[self.WP],'<',tolerance,' Tolerance met in all cells.'
      self.search=False # search complete

    if printOn: # report action
      print 'Send UAV to', WP, ' with ', "%0.3f" % (self.tmap[self.WP])

    cxxiDistanceTraveled=cxxiintegration.cxxi_send_uav_to(orders,obs,state,WP)

    # update distance traveled
    #distanceTot+= distanceLeg(platLoc,WP)
    distanceTot+=cxxiDistanceTraveled
    self.glimpses+= 1
    if printOn:
      print 'dist= ',distanceTot,' glimpse= ',glimpses

    # glimpse at location WP reduces its probability
    if updateBelief== True:
      tmap = updateBelief(WP, glimpseEffect, tmap)
    else:
      # probabilities diffuse between locations during movement since last assessment
      if printOn:
        print 'No Bayesian update'
      if Diffusion:
        diffuse(WP,colorBins)
      self.tmap[WP]= (1-glimpseEffect)*self.tmap[self.WP]
      # normalize the tmap after glimpse
      if Norm:
        normProb()

    self.platLoc=self.WP    # UAV now at goal
    if printOn: # report action
      print 'UAV clears ', self.WP,' now ', "%0.3f" % (tmap[WP])

    if glimpses>=maxGlimpses :  # Completion criteria
      print "Max glimpses reached."
      self.search= False # search complete
    if distanceTot>= maxDistance: # Completion criteria
      print "Max distance reached."
      self.search=False # search complete


# def flyRoute(platLoc, tmap):
  # ## --- Initial Conditions to execute search
  # search= True  # Start the search
  # distanceTot= 0
  # glimpses= 0

  # if behavior==1: # always fly to the greatest probability
    # print 'global greedy method'# Find next highest probability
  # if behavior==2: # not ready
    # print 'wavefront method not ready, using global greedy' # Find next highest
        probability
  # if behavior==3: # greedy with very short sight, must move to another node per glimpse
    # print 'myopic greedy method'
  # if behavior==4: # least greedy with very short sight, must move to another node per
      glimpse
    # print 'myopic trapping method'
  # if behavior==5: # shifting concentric circling for consistant wide turns
    # print 'zamboni method' # not ready
  # if behavior==6: # start in one corner, up and down rows across network
    # print 'lawnmower method' # not ready

  # # Identify first destination
  # WP= platLoc

  # ## --- Logic Loop controlling continued search
  # while search: # While search== True, continue to search
    # if behavior==1:
      # WP= greedyModule.findMaxWP(tmap) # Find next highest probability
    # if behavior==3:
      # WP= myopicGreedyModule.myopicGreedy(tmap,platLoc)
    # if behavior==4:
      # WP= myopicLeastModule.myopicLeast(tmap,platLoc,tolerance)

    # # UAV will stop if all it can see is less than acceptable tolerance
    # if tmap[WP]< tolerance :  # Completion criteria
      # print tmap[WP],'<',tolerance,' Tolerance met in all cells.'
      # search= False # search complete

    # if printOn: # report action
        # print 'Send UAV to', WP, ' with ', "%0.3f" % (tmap[WP])

    # cxxiDistanceTraveled=cxxiintegration.cxxi_send_uav_to(orders,obs,state,WP)

    # # update distance traveled
    # #distanceTot+= distanceLeg(platLoc,WP)
    # distanceTot+=cxxiDistanceTraveled
    # glimpses+= 1
    # if printOn:
      # print 'dist= ',distanceTot,' glimpse= ',glimpses

    # # glimpse at location WP reduces its probability
    # if updateBelief== True:
      # tmap = updateBelief(WP, glimpseEffect, tmap)
    # else:
      # # probabilities diffuse between locations during movement since last assessment
      # if printOn:
        # print 'No Bayesian update'
      # if Diffusion:
        # diffuse(WP,colorBins)
      # tmap[WP]= (1-glimpseEffect)*tmap[WP]
      # # normalize the tmap after glimpse
      # if Norm:
        # normProb()

    # platLoc= WP    # UAV now at goal
    # if printOn: # report action
      # print 'UAV clears ', WP,' now ', "%0.3f" % (tmap[WP])

    # if glimpses>=maxGlimpses :  # Completion criteria
      # print "Max glimpses reached."
      # search= False # search complete
    # if distanceTot>= maxDistance: # Completion criteria
      # print "Max distance reached."
      # search= False # search complete


  # print 'Return UAV to base.'
  # return distanceTot, glimpses
```

APPENDIX G. newThreatDetected.py Script

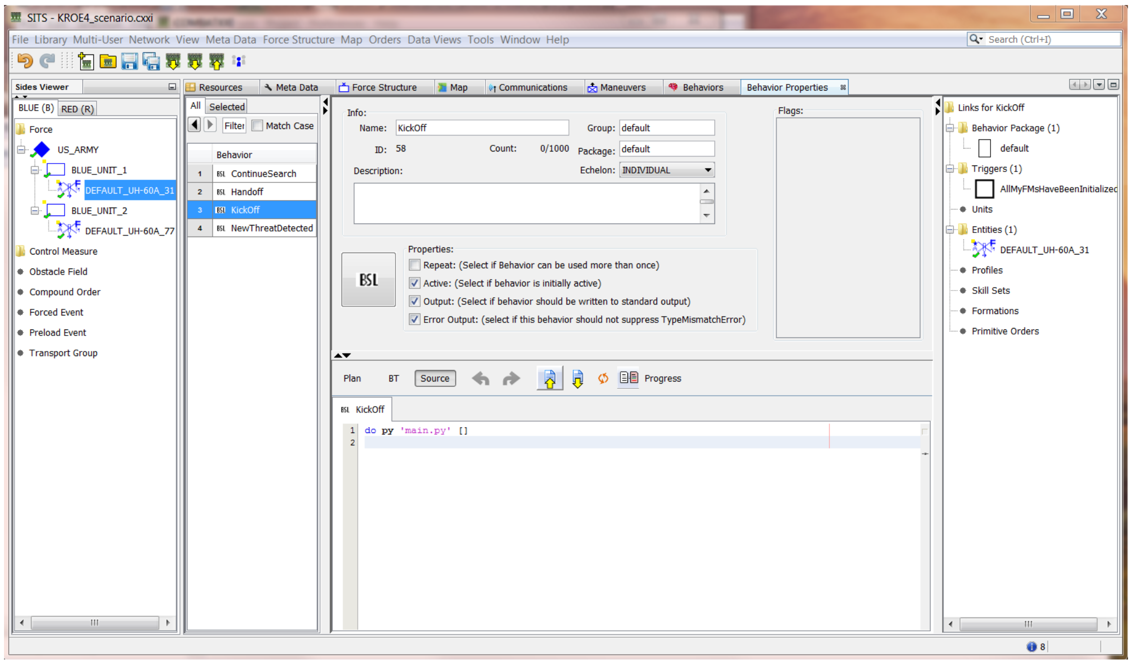
```
import Debugger
obsList=obs.getCanSee('BINOC/7X')
```

APPENDIX H. orient_sensor.py Script

```python
def straight_down(obs):
  # get handle to Human Eye sensor
  sensor=obs.getSensorByName("BINOC/7X")

  # set sensor pitch to -85 degrees.  Straight down.
  print 'Setting sensor pitch to -85 degrees'
  sensor.setViewPitch(-1.48352986)
```

APPENDIX I. SITS Behavior Property Tab Views



BLUE (B) RED (R)

- Force
  - US_ARMY
    - BLUE_UNIT_1
      - DEFAULT_UH-60A_31
    - BLUE_UNIT_2
      - DEFAULT_UH-60A_77
- Control Measure
- Obstacle Field
- Compound Order
- Forced Event
- Preload Event
- Transport Group

All | Selected | Filter | Match Case

	Behavior
1	ContinueSearch
2	Handoff
3	KickOff
4	NewThreatDetected

Info:
Name: KickOff Group: default
ID: 58 Count: 0/1000 Package: default
Description: Echelon: INDIVIDUAL

Flags:

Properties:
- [ ] Repeat: (Select if Behavior can be used more than once)
- [✓] Active: (Select if behavior is initially active)
- [✓] Output: (Select if behavior should be written to standard output)
- [✓] Error Output: (select if this behavior should not suppress TypeMismatchError)

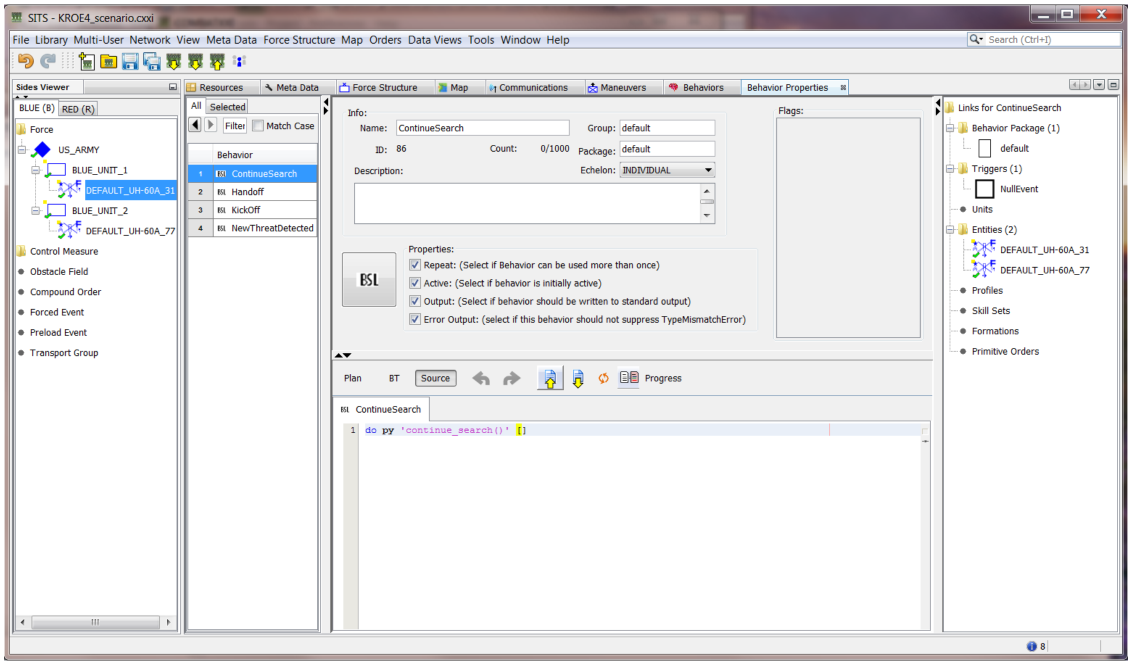
KickOff

```
do py 'main.py' []
```

Links for KickOff
- Behavior Package (1)
  - default
- Triggers (1)
  - AllMyFMsHaveBeenInitialized
- Units
- Entities (1)
  - DEFAULT_UH-60A_31
- Profiles
- Skill Sets
- Formations
- Primitive Orders

8

Figure I.1: SITS Behavior Property Tab for the behavior `KickOff`. Python script for `main.py` is found in APPENDIX D.



BLUE (B) RED (R)

- Force
  - US_ARMY
    - BLUE_UNIT_1
      - DEFAULT_UH-60A_31
    - BLUE_UNIT_2
      - DEFAULT_UH-60A_77
- Control Measure
- Obstacle Field
- Compound Order
- Forced Event
- Preload Event
- Transport Group

All | Selected | Filter | Match Case

	Behavior
1	ContinueSearch
2	Handoff
3	KickOff
4	NewThreatDetected

Info:
Name: ContinueSearch Group: default
ID: 86 Count: 0/1000 Package: default
Description: Echelon: INDIVIDUAL

Flags:

Properties:
- [✓] Repeat: (Select if Behavior can be used more than once)
- [✓] Active: (Select if behavior is initially active)
- [✓] Output: (Select if behavior should be written to standard output)
- [✓] Error Output: (select if this behavior should not suppress TypeMismatchError)

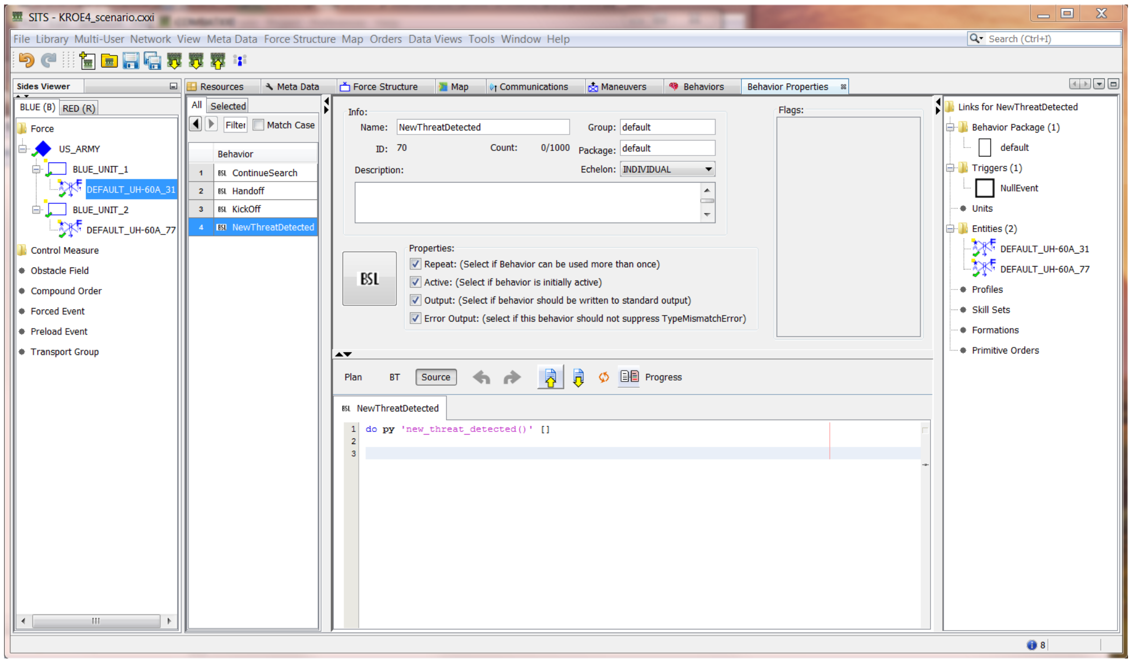
ContinueSearch

```
do py 'continue_search()' []
```

Links for ContinueSearch
- Behavior Package (1)
  - default
- Triggers (1)
  - NullEvent
- Units
- Entities (2)
  - DEFAULT_UH-60A_31
  - DEFAULT_UH-60A_77
- Profiles
- Skill Sets
- Formations
- Primitive Orders

8

Figure I.2: SITS Behavior Property Tab for the behavior `ContinueSearch`. Python script for `continue_search()` is found in APPENDIX D.



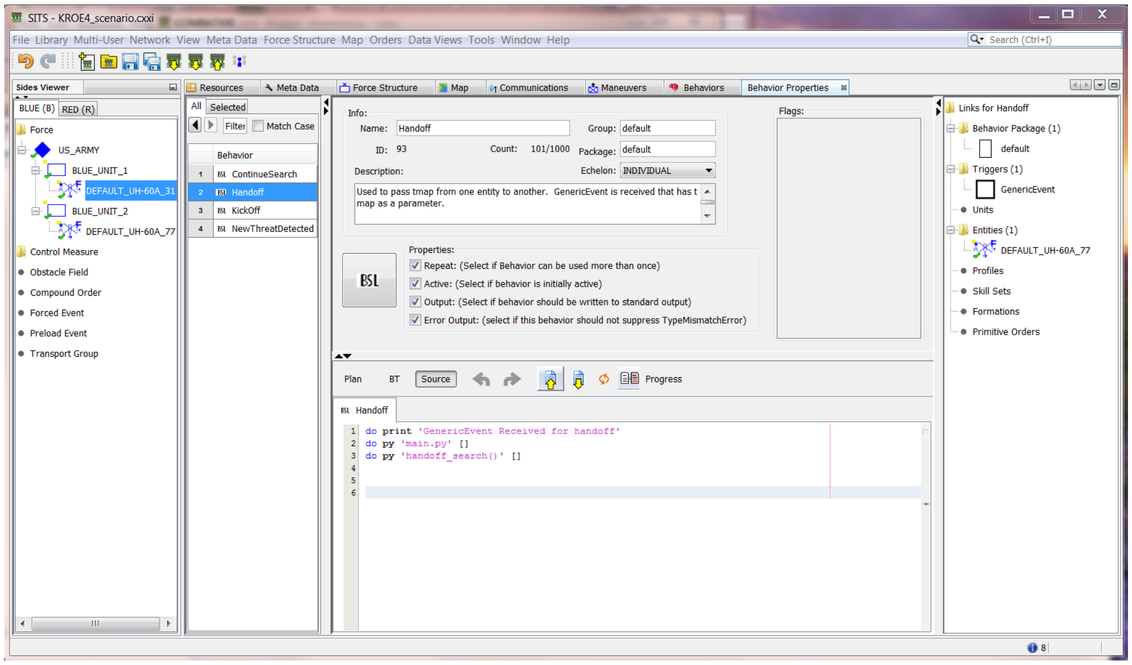
Info:
Name: NewThreatDetected Group: default
ID: 70 Count: 0/1000 Package: default
Description: Echelon: INDIVIDUAL
Flags:

Properties:
- [x] Repeat: (Select if Behavior can be used more than once)
- [x] Active: (Select if behavior is initially active)
- [x] Output: (Select if behavior should be written to standard output)
- [x] Error Output: (select if this behavior should not suppress TypeMismatchError)

NewThreatDetected

```
do py 'new_threat_detected()' []
```

Figure I.3: SITS Behavior Property Tab for the behavior `NewThreatDetected`. Python script for `new_threat_detected()` is found in APPENDIX D.



Info:
Name: Handoff Group: default
ID: 93 Count: 101/1000 Package: default
Description: Echelon: INDIVIDUAL
Used to pass tmap from one entity to another. GenericEvent is received that has t
map as a parameter.
Flags:

Properties:
- [x] Repeat: (Select if Behavior can be used more than once)
- [x] Active: (Select if behavior is initially active)
- [x] Output: (Select if behavior should be written to standard output)
- [x] Error Output: (select if this behavior should not suppress TypeMismatchError)

Handoff

```
do print 'GenericEvent Received for handoff'
do py 'main.py' []
do py 'handoff_search()' []
```

Figure I.4: SITS Behavior Property Tab for the behavior `Handoff` Python script for `handoff_search()` is found in APPENDIX D.

APPENDIX J. Existing Scenario Log Reports After Integration

```
Setting database CXXI_CONFIGURATION_DATABASE
Setting database COMMS_DATABASE
Setting database JCOMBIC_DATABASE
Setting database CXXI_MANEUVER_DATABASE
Using Script:  D:\Forge.Mil/scripts/behaviorsupport/LoadCXXISupport.py



****** Modeling Property flag setting:

FILTER_ON_SIDE = true
FILTER_DEAD_VEHICLES = true
FILTER_ON_FRIENDLY = false
FILTER_NON_ACQUIRABLE_OBJECTS = true
FILTER_NON_VIABLE_TARGETS = true
USE_TIME_LIMITED_SEARCH = true
PLAY_FALSE_TARGETS = false
USE_CUMULATIVE_KILLS_METHODOLOGY = false
TURN_JCOMBIC_ON = false
TURN_JCOMBIC_VEHICLE_DUST_ON = false
TURN_JCOMBIC_VEHICLE_SMOKE_ON = false
TURN_JCOMBIC_TANK_GUN_DUST_ON = false
MODEL_COLLISIONS = false
DETAILED_AIR_MOVE = false
DETAILED_SURFACE_MOVE = true
DETAILED_SURFACE_FORMATION_MOVE = true
MODEL_UNCONSTRAINED_HUMAN_MOVEMENT = true
MODEL_SOUNDS = false
MODEL_SOUND_DIR_ERRORS = false
MUZZLE_FLASH_DETECTION = false
MODEL_FUEL_CONSUMPTION = false
REACT_TO_INCOMING_FIRE = false
MODEL_UNCONSTRAINED_MUNITION_USAGE = false
MODEL_MUNITION_FLYBY = false
MODEL_MUNITION_SUPPRESSION_WEIGHTS = false
USE_AUTOMATIC_LASER_DESIGNATOR_SELECTION = true
ALL_AIRCRAFT_DETECT_RADAR = true
ALLOW_MUNITIONS_HITTING_UNINTENDED_TARGET = false
ALLOW_DAMAGE_EFFECTS_AGAINST_SAME_SIDE = true
INHIBIT_DAS_REPRESENTATION = false
INHIBIT_DAS_SMOKE = false
INHIBIT_GPS_JAMMER_PLAY = false
FORCE_USE_OF_DOORS_FOR_HUMANS = false
PLAY_FULL_USE_OF_DOORS = false
USE_SWE_API = false
DEFAULT_ALL_BUILDINGS_USE_SWE = false
USE_EXPLICIT_CREW = false
MODEL_TADV = false
MODEL_IMPLICIT_URBAN_CLUTTER = false
USE_ATMOSPHERIC_AREAS = false
TURN_OFF_WINDOWS = false
MODEL_PROPORTION_LOS_BLOCKED = false
MODEL_WINDOW_PROPORTION_LOS_BLOCKED = false
MODEL_ENTITIES_PROPORTION_LOS_BLOCKED = false
MODEL_BUILDING_FLOORS = false
ENTITIES_BLOCK_LOS_LOW_FIDELITY = false
ENTITIES_BLOCK_LOS_HIGH_FIDELITY = false
PERFECT_ASSOCIATION = true
STATIC_THREAT = false
USE_KALMAN_FILTER = true
USE_P_DETECT_FOR_IEDS = false
USE_ACQUIRE_TTPM = false
CLEAR_FOV_AFTER_FOR = false
CREW_MOUNTED_ENHANCED_DAMAGE_LOG = false
GENERATE_TRANSIENT_OBJECT_SHAPES = false
```

```
GENERATE_SMOKE_OBJECT_SHAPES = false
GENERATE_DUST_OBJECT_SHAPES = false
GENERATE_SENSOR_FOOTPRINT_OBJECT_SHAPES = false
ALLOW_COMBAT_MISID_CALLS = false
MODEL_TRANSIENT_OBJECT_ACQUISITION = false
MODEL_MUNITIONS_HITTING_INTERVENING_ENTITIES = false
MODEL_DAS_RESPONSE_BY_INTERVENING_ENTITIES = false
USE_DATABASE_N50_V50_VALUES = false
USE_AMSAA_POINT_DETONATING_MUNITION_METHODOLOGY = false
USE_FIRE_WHILE_MOVING_RESTRICTION = true

****** Communications Model:
USE_COMMS_BBP2 = true
USE_COMMS_CES = false

****** Development Flags:
IS_DEVELOPMENT_RUN = true
DETAILED_MOVE_UPDATE = false
INHIBIT_ALL_SEARCH_PROCESSES = false
INHIBIT_ALL_DIRECT_FIRE_ENGAGEMENTS = false
OUTPUT_CONFIGURATION_DIAGNOSTICS = false
SHARE_SCRIPT_MEMORY = true
USE_GRIDS_INSTEAD_OF_TREES = false

****** End flag values



SUCCESSFULLY LOADED
--------------------------------------------
1 BUILDING TREE(S)
0 EXTERNAL WALL TREE(S)
0 VEGETATION TREE(S)
0 URBAN AREA TREE(S)
0 WATER AREA TREE(S)
0 GENERIC AREA TREE(S)
0 VECTOR FEATURE TREE(S)
0 ATMOSPHERIC AREA TREE(S)
--------------------------------------------



**** COMBATXXI native ENV services initialized ****
Climate zone: 4
Terrain: D:\Forge.Mil\examples\terrains/Flat.trnd
BSP Tree Folder: Flat_BSPTrees
*****************************************************
Created Database Connection to jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=D:\
    Forge.Mil\examples\databases\cxxi_demo_SFF_6_0.mdb
Connection created successfully to cxxi_demo_SFF_6_0.mdb
Created Database Connection to jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=D:\
    Forge.Mil\examples\databases\commsdata_demo.mdb
Connection created successfully to commsdata_demo.mdb
Created Database Connection to jdbc:hsqldb:file:C:\Users\kyle.quinnell\.cxxi\temp/
    cxxidatabases/260629003621072_9144@wsmrwk6n5s1p1/database/database
Connection created successfully to cxxi_config.odb
Resetting CxxiPhysicalEntityTypes
Requested Air Move data for UH-60A.  Using MH-60 data.
Created Database Connection to jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=D:\
    Forge.Mil\examples\databases\cxxi_maneuver_cas.mdb
Connection created successfully to cxxi_maneuver_cas.mdb

Error connecting to Planning Database to read HBCOPRanges_Table
HBCOPRanges table not found.  Using defaults.

COP filter Radii
PLATOON : 5000.0
COMPANY : 11000.0
BATTALION : 23000.0
```

```
BRIGADE : 38000.0
DEFAULT : 5000.0

Flushing Loggers
CodePath:  D:\Forge.Mil
BuildTime:  Thu Mar 21 06:57:59 MDT 2013
Received Message: SUCCESSFULLY LOADED
Received Message: -------------------------------------------
Received Message: 1 BUILDING TREE(S)
Received Message: 0 EXTERNAL WALL TREE(S)
Received Message: 0 VEGETATION TREE(S)
Received Message: 0 URBAN AREA TREE(S)
Received Message: 0 WATER AREA TREE(S)
Received Message: 0 GENERIC AREA TREE(S)
Received Message: 0 VECTOR FEATURE TREE(S)
Received Message: 0 ATMOSPHERIC AREA TREE(S)
Received Message: -------------------------------------------
Received Message:
Received Message: DatagramSocket listening at localhost/127.0.0.1 : 4447
Using Script:  D:\Forge.Mil\examples\scenarios\modelscripts//main.py
Trigger Received:  doAllMyFMsHaveBeenInitialized
fillNormalCenter
2.5
0 0 0.00053532090306
l prob range changed to  (0.0005353209030595415, 0.0)
u prob range changed to  (0.0005353209030595415, 0.0005353209030595415)
setColors (0, 0) (0.0005353209030595415, 0.0005353209030595415)
1 0 0.108249593478
u prob range changed to  (0.0005353209030595415, 0.1082495934779059)
setColors (1, 0) (0.0005353209030595415, 0.1082495934779059)
2 0 0.798152221254
u prob range changed to  (0.0005353209030595415, 0.7981522212543952)
setColors (2, 0) (0.0005353209030595415, 0.7981522212543952)
3 0 0.108249593478
setColors (3, 0) (0.0005353209030595415, 0.7981522212543952)
4 0 0.00053532090306
setColors (4, 0) (0.0005353209030595415, 0.7981522212543952)
0 1 0.108249593478
setColors (0, 1) (0.0005353209030595415, 0.7981522212543952)
1 1 0.215963866053
setColors (1, 1) (0.0005353209030595415, 0.7981522212543952)
2 1 0.905866493829
u prob range changed to  (0.0005353209030595415, 0.9058664938292416)
setColors (2, 1) (0.0005353209030595415, 0.9058664938292416)
3 1 0.215963866053
setColors (3, 1) (0.0005353209030595415, 0.9058664938292416)
4 1 0.108249593478
setColors (4, 1) (0.0005353209030595415, 0.9058664938292416)
0 2 0.798152221254
setColors (0, 2) (0.0005353209030595415, 0.9058664938292416)
1 2 0.905866493829
setColors (1, 2) (0.0005353209030595415, 0.9058664938292416)
2 2 1.59576912161
u prob range changed to  (0.0005353209030595415, 1.5957691216057308)
setColors (2, 2) (0.0005353209030595415, 1.5957691216057308)
3 2 0.905866493829
setColors (3, 2) (0.0005353209030595415, 1.5957691216057308)
4 2 0.798152221254
setColors (4, 2) (0.0005353209030595415, 1.5957691216057308)
0 3 0.108249593478
setColors (0, 3) (0.0005353209030595415, 1.5957691216057308)
1 3 0.215963866053
setColors (1, 3) (0.0005353209030595415, 1.5957691216057308)
2 3 0.905866493829
setColors (2, 3) (0.0005353209030595415, 1.5957691216057308)
3 3 0.215963866053
setColors (3, 3) (0.0005353209030595415, 1.5957691216057308)
4 3 0.108249593478
```

```
setColors (4, 3) (0.0005353209030595415, 1.5957691216057308)
0 4 0.00053532090306
setColors (0, 4) (0.0005353209030595415, 1.5957691216057308)
1 4 0.108249593478
setColors (1, 4) (0.0005353209030595415, 1.5957691216057308)
2 4 0.798152221254
setColors (2, 4) (0.0005353209030595415, 1.5957691216057308)
3 4 0.108249593478
setColors (3, 4) (0.0005353209030595415, 1.5957691216057308)
4 4 0.00053532090306
setColors (4, 4) (0.0005353209030595415, 1.5957691216057308)
Initial threat map probabilities:
(4, 2) 0.798
(1, 4) 0.108
(3, 1) 0.216
(2, 3) 0.906
(0, 3) 0.108
(1, 3) 0.216
(4, 0) 0.001
(3, 0) 0.108
(4, 3) 0.108
(0, 1) 0.108
(4, 4) 0.001
(3, 2) 0.906
(0, 2) 0.798
(3, 4) 0.108
(1, 1) 0.216
(0, 0) 0.001
(1, 2) 0.906
(2, 2) 1.596
(4, 1) 0.108
(2, 0) 0.798
(1, 0) 0.108
(0, 4) 0.001
(2, 4) 0.798
(2, 1) 0.906
(3, 3) 0.216
initialize_search() called
global greedy method
Send UAV to (2, 2)  with  1.596
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
2 2
UAS flying to easting (GDC) Lat: 34.012859  Long: -106.234765  Elev: 1500.00
dist=  5006.24959459  glimpse=  1
No Bayesian update
start diffuse
(4, 2) 0.798
(1, 4) 0.108
(3, 1) 0.216
(2, 3) 0.906
(0, 3) 0.108
(1, 3) 0.216
(4, 0) 0.001
(3, 0) 0.108
(4, 3) 0.108
(0, 1) 0.108
(4, 4) 0.001
(3, 2) 0.906
(0, 2) 0.798
(3, 4) 0.108
(1, 1) 0.216
(0, 0) 0.001
(1, 2) 0.906
(2, 2) 1.596
(4, 1) 0.108
(2, 0) 0.798
(1, 0) 0.108
```

```
(0, 4) 0.001
(2, 4) 0.798
(2, 1) 0.906
(3, 3) 0.216
diffuse complete
(4, 2) 0.799
(1, 4) 0.113
(3, 1) 0.221
(2, 3) 0.907
(0, 3) 0.109
(1, 3) 0.221
(4, 0) 0.001
(3, 0) 0.109
(4, 3) 0.108
(0, 1) 0.114
(4, 4) 0.001
(3, 2) 0.907
(0, 2) 0.800
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.002
(1, 2) 0.912
(2, 2) 1.597
(4, 1) 0.113
(2, 0) 0.800
(1, 0) 0.114
(0, 4) 0.001
(2, 4) 0.799
(2, 1) 0.912
(3, 3) 0.216
UAV clears  (2, 2)  now  0.000
Threat detected
Send UAV to (2, 1)  with  0.912
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
2 1
UAS flying to easting (GDC) Lat: 34.021875  Long: -106.234896  Elev: 1500.00
dist=  6006.72548441  glimpse=  2
No Bayesian update
start diffuse
(4, 2) 0.799
(1, 4) 0.113
(3, 1) 0.221
(2, 3) 0.907
(0, 3) 0.109
(1, 3) 0.221
(4, 0) 0.001
(3, 0) 0.109
(4, 3) 0.108
(0, 1) 0.114
(4, 4) 0.001
(3, 2) 0.907
(0, 2) 0.800
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.002
(1, 2) 0.912
(2, 2) 0.000
(4, 1) 0.113
(2, 0) 0.800
(1, 0) 0.114
(0, 4) 0.001
(2, 4) 0.799
(2, 1) 0.912
(3, 3) 0.216
diffuse complete
(4, 2) 0.800
(1, 4) 0.118
```

```
(3, 1) 0.226
(2, 3) 0.907
(0, 3) 0.110
(1, 3) 0.226
(4, 0) 0.002
(3, 0) 0.110
(4, 3) 0.108
(0, 1) 0.120
(4, 4) 0.002
(3, 2) 0.907
(0, 2) 0.801
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.003
(1, 2) 0.917
(2, 2) 0.001
(4, 1) 0.118
(2, 0) 0.801
(1, 0) 0.120
(0, 4) 0.002
(2, 4) 0.800
(2, 1) 0.917
(3, 3) 0.216
UAV clears  (2, 1)  now  0.000
Threat not detected
Send UAV to (1, 2)  with  0.917
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
1 2
UAS flying to easting (GDC) Lat: 34.012749  Long: -106.245593  Elev: 1500.00
dist=  7421.60982581  glimpse=  3
No Bayesian update
start diffuse
(4, 2) 0.800
(1, 4) 0.118
(3, 1) 0.226
(2, 3) 0.907
(0, 3) 0.110
(1, 3) 0.226
(4, 0) 0.002
(3, 0) 0.110
(4, 3) 0.108
(0, 1) 0.120
(4, 4) 0.002
(3, 2) 0.907
(0, 2) 0.801
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.003
(1, 2) 0.917
(2, 2) 0.001
(4, 1) 0.118
(2, 0) 0.801
(1, 0) 0.120
(0, 4) 0.002
(2, 4) 0.800
(2, 1) 0.000
(3, 3) 0.216
diffuse complete
(4, 2) 0.800
(1, 4) 0.123
(3, 1) 0.231
(2, 3) 0.908
(0, 3) 0.110
(1, 3) 0.231
(4, 0) 0.003
(3, 0) 0.110
(4, 3) 0.108
```

```
(0, 1) 0.125
(4, 4) 0.002
(3, 2) 0.908
(0, 2) 0.802
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.005
(1, 2) 0.923
(2, 2) 0.003
(4, 1) 0.123
(2, 0) 0.802
(1, 0) 0.120
(0, 4) 0.003
(2, 4) 0.800
(2, 1) 0.006
(3, 3) 0.216
UAV clears  (1, 2)  now  0.000
Threat detected
Send UAV to (3, 2)  with  0.908
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
3 2
UAS flying to easting (GDC) Lat: 34.012967  Long: -106.223937  Elev: 1500.00
dist=  9422.56089297  glimpse=  4
No Bayesian update
start diffuse
(4, 2) 0.800
(1, 4) 0.123
(3, 1) 0.231
(2, 3) 0.908
(0, 3) 0.110
(1, 3) 0.231
(4, 0) 0.003
(3, 0) 0.110
(4, 3) 0.108
(0, 1) 0.125
(4, 4) 0.002
(3, 2) 0.908
(0, 2) 0.802
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.005
(1, 2) 0.000
(2, 2) 0.003
(4, 1) 0.123
(2, 0) 0.802
(1, 0) 0.120
(0, 4) 0.003
(2, 4) 0.800
(2, 1) 0.006
(3, 3) 0.216
diffuse complete
(4, 2) 0.801
(1, 4) 0.128
(3, 1) 0.236
(2, 3) 0.909
(0, 3) 0.111
(1, 3) 0.236
(4, 0) 0.003
(3, 0) 0.111
(4, 3) 0.108
(0, 1) 0.125
(4, 4) 0.003
(3, 2) 0.909
(0, 2) 0.804
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.006
```

```
(1, 2) 0.006
(2, 2) 0.004
(4, 1) 0.128
(2, 0) 0.804
(1, 0) 0.120
(0, 4) 0.004
(2, 4) 0.801
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (3, 2)  now  0.000
Threat not detected
Send UAV to (2, 3)  with  0.909
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
2 3
UAS flying to easting (GDC) Lat: 34.003842  Long: -106.234635  Elev: 1500.00
dist=  10837.4492035  glimpse=  5
No Bayesian update
start diffuse
(4, 2) 0.801
(1, 4) 0.128
(3, 1) 0.236
(2, 3) 0.909
(0, 3) 0.111
(1, 3) 0.236
(4, 0) 0.003
(3, 0) 0.111
(4, 3) 0.108
(0, 1) 0.125
(4, 4) 0.003
(3, 2) 0.000
(0, 2) 0.804
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.006
(1, 2) 0.006
(2, 2) 0.004
(4, 1) 0.128
(2, 0) 0.804
(1, 0) 0.120
(0, 4) 0.004
(2, 4) 0.801
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.802
(1, 4) 0.133
(3, 1) 0.241
(2, 3) 0.909
(0, 3) 0.112
(1, 3) 0.241
(4, 0) 0.004
(3, 0) 0.112
(4, 3) 0.108
(0, 1) 0.125
(4, 4) 0.003
(3, 2) 0.001
(0, 2) 0.805
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.008
(1, 2) 0.011
(2, 2) 0.005
(4, 1) 0.133
(2, 0) 0.805
(1, 0) 0.120
(0, 4) 0.004
(2, 4) 0.802
```

```
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (2, 3)  now  0.000
Threat detected
Send UAV to (2, 0)  with  0.805
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
2 0
UAS flying to easting (GDC) Lat: 34.030892  Long: -106.235026  Elev: 1500.00
dist=  13838.8768482  glimpse=  6
No Bayesian update
start diffuse
(4, 2) 0.802
(1, 4) 0.133
(3, 1) 0.241
(2, 3) 0.000
(0, 3) 0.112
(1, 3) 0.241
(4, 0) 0.004
(3, 0) 0.112
(4, 3) 0.108
(0, 1) 0.125
(4, 4) 0.003
(3, 2) 0.001
(0, 2) 0.805
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.008
(1, 2) 0.011
(2, 2) 0.005
(4, 1) 0.133
(2, 0) 0.805
(1, 0) 0.120
(0, 4) 0.004
(2, 4) 0.802
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.802
(1, 4) 0.138
(3, 1) 0.246
(2, 3) 0.001
(0, 3) 0.113
(1, 3) 0.246
(4, 0) 0.005
(3, 0) 0.113
(4, 3) 0.108
(0, 1) 0.125
(4, 4) 0.004
(3, 2) 0.001
(0, 2) 0.807
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.009
(1, 2) 0.011
(2, 2) 0.007
(4, 1) 0.138
(2, 0) 0.807
(1, 0) 0.120
(0, 4) 0.005
(2, 4) 0.802
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (2, 0)  now  0.000
Threat not detected
Send UAV to (0, 2)  with  0.807
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
```

```
0 2
UAS flying to easting (GDC) Lat: 34.012639  Long: -106.256420  Elev: 1500.00
dist=  16668.6415096  glimpse=  7
No Bayesian update
start diffuse
(4, 2) 0.802
(1, 4) 0.138
(3, 1) 0.246
(2, 3) 0.001
(0, 3) 0.113
(1, 3) 0.246
(4, 0) 0.005
(3, 0) 0.113
(4, 3) 0.108
(0, 1) 0.125
(4, 4) 0.004
(3, 2) 0.001
(0, 2) 0.807
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.009
(1, 2) 0.011
(2, 2) 0.007
(4, 1) 0.138
(2, 0) 0.000
(1, 0) 0.120
(0, 4) 0.005
(2, 4) 0.802
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.803
(1, 4) 0.143
(3, 1) 0.251
(2, 3) 0.001
(0, 3) 0.114
(1, 3) 0.251
(4, 0) 0.006
(3, 0) 0.114
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.004
(3, 2) 0.002
(0, 2) 0.808
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.010
(1, 2) 0.011
(2, 2) 0.008
(4, 1) 0.143
(2, 0) 0.002
(1, 0) 0.120
(0, 4) 0.006
(2, 4) 0.803
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (0, 2)  now  0.000
Threat detected
Send UAV to (2, 4)  with  0.803
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
2 4
UAS flying to easting (GDC) Lat: 33.994825  Long: -106.234504  Elev: 1500.00
dist=  19498.4061401  glimpse=  8
No Bayesian update
start diffuse
(4, 2) 0.803
(1, 4) 0.143
```

```
(3, 1) 0.251
(2, 3) 0.001
(0, 3) 0.114
(1, 3) 0.251
(4, 0) 0.006
(3, 0) 0.114
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.004
(3, 2) 0.002
(0, 2) 0.000
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.010
(1, 2) 0.011
(2, 2) 0.008
(4, 1) 0.143
(2, 0) 0.002
(1, 0) 0.120
(0, 4) 0.006
(2, 4) 0.803
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.804
(1, 4) 0.148
(3, 1) 0.256
(2, 3) 0.002
(0, 3) 0.115
(1, 3) 0.256
(4, 0) 0.007
(3, 0) 0.115
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.005
(3, 2) 0.003
(0, 2) 0.002
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.012
(1, 2) 0.011
(2, 2) 0.009
(4, 1) 0.148
(2, 0) 0.003
(1, 0) 0.120
(0, 4) 0.007
(2, 4) 0.804
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (2, 4)  now  0.000
Threat detected
Send UAV to (4, 2)  with  0.804
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
4 2
UAS flying to easting (GDC) Lat: 34.013074  Long: -106.213110  Elev: 1500.00
dist=  22328.1866782  glimpse=  9
No Bayesian update
start diffuse
(4, 2) 0.804
(1, 4) 0.148
(3, 1) 0.256
(2, 3) 0.002
(0, 3) 0.115
(1, 3) 0.256
(4, 0) 0.007
(3, 0) 0.115
(4, 3) 0.108
```

```
(0, 1) 0.126
(4, 4) 0.005
(3, 2) 0.003
(0, 2) 0.002
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.012
(1, 2) 0.011
(2, 2) 0.009
(4, 1) 0.148
(2, 0) 0.003
(1, 0) 0.120
(0, 4) 0.007
(2, 4) 0.000
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.804
(1, 4) 0.148
(3, 1) 0.261
(2, 3) 0.003
(0, 3) 0.116
(1, 3) 0.256
(4, 0) 0.008
(3, 0) 0.115
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.005
(3, 2) 0.003
(0, 2) 0.003
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.013
(1, 2) 0.011
(2, 2) 0.011
(4, 1) 0.153
(2, 0) 0.005
(1, 0) 0.120
(0, 4) 0.008
(2, 4) 0.001
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (4, 2)  now  0.000
Threat not detected
Send UAV to (3, 1)  with  0.261
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
3 1
UAS flying to easting (GDC) Lat: 34.021984  Long: -106.224067  Elev: 1500.00
dist=  23743.0789071  glimpse=  10
No Bayesian update
start diffuse
(4, 2) 0.000
(1, 4) 0.148
(3, 1) 0.261
(2, 3) 0.003
(0, 3) 0.116
(1, 3) 0.256
(4, 0) 0.008
(3, 0) 0.115
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.005
(3, 2) 0.003
(0, 2) 0.003
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.013
```

```
(1, 2) 0.011
(2, 2) 0.011
(4, 1) 0.153
(2, 0) 0.005
(1, 0) 0.120
(0, 4) 0.008
(2, 4) 0.001
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.001
(1, 4) 0.148
(3, 1) 0.261
(2, 3) 0.003
(0, 3) 0.117
(1, 3) 0.256
(4, 0) 0.009
(3, 0) 0.116
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.005
(3, 2) 0.004
(0, 2) 0.005
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.015
(1, 2) 0.011
(2, 2) 0.012
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.009
(2, 4) 0.001
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (3, 1)  now  0.000
Threat detected
Send UAV to (1, 3)  with  0.256
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
1 3
UAS flying to easting (GDC) Lat: 34.003733  Long: -106.245461  Elev: 1500.00
dist=  26572.8515416  glimpse=  11
No Bayesian update
start diffuse
(4, 2) 0.001
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.003
(0, 3) 0.117
(1, 3) 0.256
(4, 0) 0.009
(3, 0) 0.116
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.005
(3, 2) 0.004
(0, 2) 0.005
(3, 4) 0.108
(1, 1) 0.226
(0, 0) 0.015
(1, 2) 0.011
(2, 2) 0.012
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.009
(2, 4) 0.001
```

```
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.001
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.004
(0, 3) 0.118
(1, 3) 0.256
(4, 0) 0.010
(3, 0) 0.117
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.006
(3, 2) 0.005
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.226
(0, 0) 0.016
(1, 2) 0.011
(2, 2) 0.014
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.010
(2, 4) 0.002
(2, 1) 0.011
(3, 3) 0.216
UAV clears  (1, 3)  now  0.000
Threat not detected
Send UAV to (1, 1)  with  0.226
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
1 1
UAS flying to easting (GDC) Lat: 34.021766  Long: -106.245725  Elev: 1500.00
dist=  28573.7976644  glimpse=  12
No Bayesian update
start diffuse
(4, 2) 0.001
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.004
(0, 3) 0.118
(1, 3) 0.000
(4, 0) 0.010
(3, 0) 0.117
(4, 3) 0.108
(0, 1) 0.126
(4, 4) 0.006
(3, 2) 0.005
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.226
(0, 0) 0.016
(1, 2) 0.011
(2, 2) 0.014
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.010
(2, 4) 0.002
(2, 1) 0.011
(3, 3) 0.216
diffuse complete
(4, 2) 0.002
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.005
```

```
(0, 3) 0.118
(1, 3) 0.000
(4, 0) 0.011
(3, 0) 0.118
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.006
(3, 2) 0.005
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.226
(0, 0) 0.017
(1, 2) 0.011
(2, 2) 0.015
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.011
(2, 4) 0.003
(2, 1) 0.012
(3, 3) 0.216
UAV clears  (1, 1)  now  0.000
Threat detected
Send UAV to (3, 3)  with  0.216
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
3 3
UAS flying to easting (GDC) Lat: 34.003950  Long: -106.223808  Elev: 1500.00
dist=  31403.5702675  glimpse=  13
No Bayesian update
start diffuse
(4, 2) 0.002
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.005
(0, 3) 0.118
(1, 3) 0.000
(4, 0) 0.011
(3, 0) 0.118
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.006
(3, 2) 0.005
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.011
(2, 2) 0.015
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.011
(2, 4) 0.003
(2, 1) 0.012
(3, 3) 0.216
diffuse complete
(4, 2) 0.003
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.005
(0, 3) 0.119
(1, 3) 0.000
(4, 0) 0.012
(3, 0) 0.119
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.007
```

```
(3, 2) 0.006
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.012
(2, 4) 0.003
(2, 1) 0.012
(3, 3) 0.216
UAV clears  (3, 3)  now  0.000
Max distance reached.
Return UAV to base.
Threat not detected
Parking
Search Complete.  Handing off to entity DEFAULT_UH-60A_77
BSL 7040.3570 DEFAULT_UH-60A_77>GenericEvent Received for handoff
Trigger Received:  doGenericEvent
fillNormalCenter
2.5
0 0 0.00053532090306
l prob range changed to  (0.000535320903059541 5, 0.0)
u prob range changed to  (0.000535320903059541 5, 0.000535320903059541 5)
setColors (0, 0) (0.000535320903059541 5, 0.000535320903059541 5)
1 0 0.108249593478
u prob range changed to  (0.000535320903059541 5, 0.1082495934779059)
setColors (1, 0) (0.000535320903059541 5, 0.1082495934779059)
2 0 0.798152221254
u prob range changed to  (0.000535320903059541 5, 0.7981522212543952)
setColors (2, 0) (0.000535320903059541 5, 0.7981522212543952)
3 0 0.108249593478
setColors (3, 0) (0.000535320903059541 5, 0.7981522212543952)
4 0 0.00053532090306
setColors (4, 0) (0.000535320903059541 5, 0.7981522212543952)
0 1 0.108249593478
setColors (0, 1) (0.000535320903059541 5, 0.7981522212543952)
1 1 0.215963866053
setColors (1, 1) (0.000535320903059541 5, 0.7981522212543952)
2 1 0.905866493829
u prob range changed to  (0.000535320903059541 5, 0.9058664938292416)
setColors (2, 1) (0.000535320903059541 5, 0.9058664938292416)
3 1 0.215963866053
setColors (3, 1) (0.000535320903059541 5, 0.9058664938292416)
4 1 0.108249593478
setColors (4, 1) (0.000535320903059541 5, 0.9058664938292416)
0 2 0.798152221254
setColors (0, 2) (0.000535320903059541 5, 0.9058664938292416)
1 2 0.905866493829
setColors (1, 2) (0.000535320903059541 5, 0.9058664938292416)
2 2 1.59576912161
u prob range changed to  (0.000535320903059541 5, 1.5957691216057308)
setColors (2, 2) (0.000535320903059541 5, 1.5957691216057308)
3 2 0.905866493829
setColors (3, 2) (0.000535320903059541 5, 1.5957691216057308)
4 2 0.798152221254
setColors (4, 2) (0.000535320903059541 5, 1.5957691216057308)
0 3 0.108249593478
setColors (0, 3) (0.000535320903059541 5, 1.5957691216057308)
1 3 0.215963866053
setColors (1, 3) (0.000535320903059541 5, 1.5957691216057308)
2 3 0.905866493829
setColors (2, 3) (0.000535320903059541 5, 1.5957691216057308)
3 3 0.215963866053
setColors (3, 3) (0.000535320903059541 5, 1.5957691216057308)
```

```
4 3 0.108249593478
setColors (4, 3) (0.0005353209030595415, 1.5957691216057308)
0 4 0.00053532090306
setColors (0, 4) (0.0005353209030595415, 1.5957691216057308)
1 4 0.108249593478
setColors (1, 4) (0.0005353209030595415, 1.5957691216057308)
2 4 0.798152221254
setColors (2, 4) (0.0005353209030595415, 1.5957691216057308)
3 4 0.108249593478
setColors (3, 4) (0.0005353209030595415, 1.5957691216057308)
4 4 0.00053532090306
setColors (4, 4) (0.0005353209030595415, 1.5957691216057308)
Initial threat map probabilities:
(4, 2) 0.798
(1, 4) 0.108
(3, 1) 0.216
(2, 3) 0.906
(0, 3) 0.108
(1, 3) 0.216
(4, 0) 0.001
(3, 0) 0.108
(4, 3) 0.108
(0, 1) 0.108
(4, 4) 0.001
(3, 2) 0.906
(0, 2) 0.798
(3, 4) 0.108
(1, 1) 0.216
(0, 0) 0.001
(1, 2) 0.906
(2, 2) 1.596
(4, 1) 0.108
(2, 0) 0.798
(1, 0) 0.108
(0, 4) 0.001
(2, 4) 0.798
(2, 1) 0.906
(3, 3) 0.216
initialize_search() called
global greedy method
Send UAV to (4, 1)  with  0.153
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
4 1
UAS flying to easting (GDC) Lat: 34.022091  Long: -106.213238  Elev: 1500.00
dist=  6379.30871669  glimpse=  1
No Bayesian update
start diffuse
(4, 2) 0.003
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.005
(0, 3) 0.119
(1, 3) 0.000
(4, 0) 0.012
(3, 0) 0.119
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.007
(3, 2) 0.006
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.153
(2, 0) 0.006
```

```
(1, 0) 0.120
(0, 4) 0.012
(2, 4) 0.003
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.003
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.006
(0, 3) 0.120
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.007
(3, 2) 0.007
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.153
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.013
(2, 4) 0.004
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (4, 1)  now  0.000
Threat not detected
Send UAV to (1, 4)  with  0.148
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
1 4
UAS flying to easting (GDC) Lat: 33.994716  Long: -106.245330  Elev: 1500.00
dist=  10623.9735438  glimpse=  2
No Bayesian update
start diffuse
(4, 2) 0.003
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.006
(0, 3) 0.120
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.007
(3, 2) 0.007
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.013
(2, 4) 0.004
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.004
```

```
(1, 4) 0.148
(3, 1) 0.000
(2, 3) 0.007
(0, 3) 0.121
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.007
(3, 2) 0.007
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.014
(2, 4) 0.005
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (1, 4)  now  0.000
Threat detected
Send UAV to (0, 1)  with  0.126
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
0 1
UAS flying to easting (GDC) Lat: 34.021656  Long: -106.256553  Elev: 1500.00
dist=  13787.7425217  glimpse=  3
No Bayesian update
start diffuse
(4, 2) 0.004
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.007
(0, 3) 0.121
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.007
(3, 2) 0.007
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.120
(0, 4) 0.014
(2, 4) 0.005
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.005
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.007
(0, 3) 0.121
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
```

```
(4, 3) 0.109
(0, 1) 0.126
(4, 4) 0.007
(3, 2) 0.008
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.121
(0, 4) 0.014
(2, 4) 0.005
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (0, 1)  now  0.000
Threat not detected
Send UAV to (0, 3)  with  0.121
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
0 3
UAS flying to easting (GDC) Lat: 34.003623  Long: -106.256288  Elev: 1500.00
dist=  15788.6829445  glimpse=  4
No Bayesian update
start diffuse
(4, 2) 0.005
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.007
(0, 3) 0.121
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.007
(3, 2) 0.008
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.017
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.121
(0, 4) 0.014
(2, 4) 0.005
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.005
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.008
(0, 3) 0.121
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.009
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
```

```
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.121
(0, 4) 0.014
(2, 4) 0.006
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (0, 3)  now  0.000
Threat detected
Send UAV to (1, 0)  with  0.121
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
1 0
UAS flying to easting (GDC) Lat: 34.030783  Long: -106.245856  Elev: 1500.00
dist=  18952.4519428  glimpse=  5
No Bayesian update
start diffuse
(4, 2) 0.005
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.008
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.009
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.000
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.121
(0, 4) 0.014
(2, 4) 0.006
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.006
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.009
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.009
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.121
(0, 4) 0.014
```

```
(2, 4) 0.007
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (1, 0)  now  0.000
Threat not detected
Send UAV to (3, 0)  with  0.120
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
3 0
UAS flying to easting (GDC) Lat: 34.031000  Long: -106.224196  Elev: 1500.00
dist=  20953.4030101  glimpse=  6
No Bayesian update
start diffuse
(4, 2) 0.006
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.009
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.009
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.007
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.007
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.009
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.120
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.010
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.007
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (3, 0)  now  0.000
Threat not detected
Send UAV to (3, 4)  with  0.109
Setting sensor pitch to -85 degrees
```

```
Calling cxxi_send_uav_to
3 4
UAS flying to easting (GDC) Lat: 33.994933  Long: -106.223679  Elev: 1500.00
dist=  24955.3177103  glimpse=  7
No Bayesian update
start diffuse
(4, 2) 0.007
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.009
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.000
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.010
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.007
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.007
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.010
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.000
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.011
(0, 2) 0.006
(3, 4) 0.109
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.008
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (3, 4)  now  0.000
Threat not detected
Send UAV to (4, 3)  with  0.109
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
4 3
UAS flying to easting (GDC) Lat: 34.004057  Long: -106.212981  Elev: 1500.00
dist=  26370.2099552  glimpse=  8
No Bayesian update
start diffuse
(4, 2) 0.007
```

```
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.010
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.000
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.011
(0, 2) 0.006
(3, 4) 0.000
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.008
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.008
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.010
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.000
(4, 3) 0.109
(0, 1) 0.000
(4, 4) 0.009
(3, 2) 0.012
(0, 2) 0.006
(3, 4) 0.000
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.008
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (4, 3)  now  0.000
Threat not detected
Send UAV to (0, 0)  with  0.018
Setting sensor pitch to -85 degrees
Calling cxxi_send_uav_to
0 0
UAS flying to easting (GDC) Lat: 34.030673  Long: -106.256686  Elev: 1500.00
dist=  31372.5880556  glimpse=  9
No Bayesian update
start diffuse
(4, 2) 0.008
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.010
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.000
```

```
(4, 3) 0.000
(0, 1) 0.000
(4, 4) 0.009
(3, 2) 0.012
(0, 2) 0.006
(3, 4) 0.000
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.008
(2, 1) 0.012
(3, 3) 0.000
diffuse complete
(4, 2) 0.008
(1, 4) 0.000
(3, 1) 0.000
(2, 3) 0.010
(0, 3) 0.000
(1, 3) 0.000
(4, 0) 0.013
(3, 0) 0.000
(4, 3) 0.000
(0, 1) 0.000
(4, 4) 0.008
(3, 2) 0.012
(0, 2) 0.006
(3, 4) 0.000
(1, 1) 0.001
(0, 0) 0.018
(1, 2) 0.012
(2, 2) 0.016
(4, 1) 0.000
(2, 0) 0.006
(1, 0) 0.000
(0, 4) 0.014
(2, 4) 0.008
(2, 1) 0.012
(3, 3) 0.000
UAV clears  (0, 0)  now  0.000
Max distance reached.
Return UAV to base.
Threat not detected
Parking
Search Complete.  Handing off to entity DEFAULT_UH-60A_77
```

REFERENCES

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice Hall, Upper Saddle River, NJ, 1993.

[2] LTC Jonathan Alt. Initial FY13 Research Plan. Technical report, TRADOC Analysis Center, Monterey, 700 Dyer Road, Monterey, CA 93943-0692, 2013.

[3] COMBATXXI ATRC-WM. COMBATXXI release candidate 2.2. 2013.

[4] Rene G Burgess and Christian R.Darken. Realistic Human Path Planning using Fluid Simulation. *Proceedings of Behavior Representation in Modeling and Simulation*, pages 145–162, 2004.

[5] Kleijnen J. P. C. "Theory and Methodology Verification and Validation of Simulation Models". *European Journal of Operational Research*, 82:145–162, 1995.

[6] Lance E. Champagne. Development Approaches Coupled with Verification and Validation Methodologies for Agent-Based Mission-Level Analytical Combat Simulations, March 2004.

[7] Drusinsky D., James B. Michael, and Mantak Shing. The Three Dimensions of Formal Validation and Verification of Reactive System Behaviors. Technical Report NPS-CS-07-008, Naval Postgraduate School Monterey, CA 93943-5000, August 2007.

[8] James N Elele. Assessing risk levels of verification, validation, and accreditation of models and simulations. In *SPIE Defense, Security, and Sensing*, pages 73480C–73480C. International Society for Optics and Photonics, 2009.

[9] Naomi Hinojosa. COMBATXXI Behaviors https://hq.tradoc.army.mil/sites/trac/hq/ecp/lists/tools/. 2013.

[10] MAJ Christopher Marks. Mission Command Analysis Using Monte Carlo Tree Search. Technical Report TRAC-M-TR-13-050, TRADOC Analysis Center, Monterey, 700 Dyer Road, Monterey, CA 93943-0692, June 2013.

[11] MAJ Edward Masotti. Support to Deployed Force Protection. Technical Report TRAC-M-TR-13-019, TRADOC Analysis Center, Monterey, 700 Dyer Road, Monterey, CA 93943-0692, 2013.

[12] Chris Morey. Methodology Slide Development. TRADOC Analysis Center, September 2005.

[13] Rich Pace, Mike Hannon, Dave Ohman, and Mario Torres. COMBATXXI description. https://hq.tradoc.army.mil/sites/trac/hq/ecp/Lists/Tools/, 2013.

[14] Steve Rabin. *AI Game Programming Wisdom 2*. Charles River Media, Inc., Hingham, MA, 2003.

[15] TRAC-White Sands Missile Range. COMBATXXI users guide. 2012.

[16] TRADOC Analysis Center. Constraints, Limitations, and Assumptions, Code of Best Practice. Technical Report TRAC-H-TM-12-033, 255 Sedgwick Avenue, Fort Leavenworth, KS 66027-2345, June 2012.