A Graphical Interface for Managing Multiple Unmanned Aerial Vehicles

Charles J. Cohen, Ph.D. & Ron Hay
Cybernet Systems Corporation
Ann Arbor, MI 48108
ccohen@cybernet.com, rhay@cybernet.com

ABSTRACT

The control and/or command of multiple Uninhabited Air Vehicles (UAVs) within an actual or simulated operational theater requires an interface that allows the user to direct the actions of these UAVs in real-time. The user must, at the very least, have the capability to issue commands that apply to whole UAV groups ("swarms"), as well as the ability to narrow commands to a few or even a single UAV. The user must have the ability to create UAV groups, select and assign targets, and interactively generate and assign routes.

This paper describes an innovative interface for operators to control multiple UAVs in a combat situation. Coupled with the latest off-the-shelf input and output hardware, the software will be an intuitive, real-time, graphical, 3D environment that enhances situation awareness as much as possible without perceptual overload. This system will also be fully compatible with the Mixed Initiative Control for Automa-teams (MICA) Open Experimental Platform (OEP) multiple UAV scenario system.

To control multiple UAVs, there are specific command requirements for the UAVMI system. Relevant issues include:

1. Where and how exactly does the UAV's autonomy interface with the higher level commands coming from the UAVMI? The solution is to create (or adopt an existing) command language for the individual UAVs.
2. Define a library of "swarm-level commands". These are commands that affect groups of UAVs.
3. Address the issue of how "swarm-level commands" translate into commands to individual UAVs. For example: if the swarm-level command is for *group A* to attack enemy *group E*, then what exactly should a_1 (one UAV in *group A*) do? Attack e_1? Attack the closest element in *group E*? What if that target is already covered by a_3?

The areas addressed include group and individual commands, path generation, relative posture, selection methods, viewpoint navigation, and information displays.

ABOUT THE AUTHORS

Dr. Charles J. Cohen has been working in the fields of image processing, robotics, human-computer interaction, and artificial intelligence for over a decade. At Cybernet, he has been the project manager for many projects for the United States Armed Forces (Air Force, Navy, and Army), National Aeronautics and Space Administration, and other government agencies. His projects include work on simulation, training, real-time optical pose determination, robotics, virtual reality, object identification, feature and body tracking, and human performance evaluation.
Dr. Cohen's main areas of interest is gesture recognition and massive number of agent network architectures and systems, which he is developing and integrating for the Army and Air Force.

Mr. Ron Hay is a lead developer in Cybernet's Virtual Reality group, specializing in graphics programming (using both OpenGL and DirectX). The core technology for the Virtual Reality group is the OpenSkies Simulation System. OpenSkies has been through several iterations of development over the past four years, and includes a complete, standalone HLA implementation of great sophistication, standalone components for panel development, physics simulation, scenegraph API, scenario development and deployment, and collision detection. It has been used for everything from a first-year Naval pilot trainer, to a massive-multiplayer game, to an astronomy education tool. Mr. Hay has acted as one of the primary developers for the system since its birth.

A Graphical Interface for Managing Multiple Unmanned Aerial Vehicles

Charles J. Cohen, Ph.D. & Ron Hay
Cybernet Systems Corporation
Ann Arbor, MI 48108
ccohen@cybernet.com, rhay@cybernet.com

BACKGROUND

Typical interfaces for controlling multiple UAVs have focused on surveillance applications or longer-term mission planning. Such interfaces are not sufficiently real-time for implementing short-term combat tactics and decision-making (see Francis, 2003). Furthermore, these interfaces are similar to paper-based mission planning approaches in that they are inherently 2-dimensional. This is a serious shortcoming when the autonomous forces being commanded are airborne and require 3-dimensions of direction, especially with regards to data overload and information camouflage (see Stark, 2003).

We are developing an Uninhabited Air Vehicle Management Interface (UAVMI) system centered on a 3-dimensional graphical representation of the geographic space. The operator is able to move a 3D cursor around within a geographic volume, selecting groups, paths, and targets in 3D space. While other swarm interfaces deal mainly with abstracted virtual agents (such as Minar et. al, 1996 and Tambe et. al, 1999), this interface allows the user to essentially reach into the virtual battlefield/airspace to control swarms of UAV entities.

The UAVMI will be constructed as an interface/library that can be attached to existing simulation/command software. Typically this interface is constructed as a software module that is linked into the specific simulation software. This approach requires that the simulation software be altered somewhat in order to communicate with the UAVMI. The High Level Architecture (HLA) (see Larimer, 1997) provides another solution that could be used as an alternate method for communicating commands to the UAVs. Using HLA, a computer dedicated to the UAVMI system could be placed on a network that includes multiple machines participating in the simulation. The UAVMI system would issue commands, through HLA, to the various UAVs being controlled/simulated on other participating machines. Design and development of the UAVMI system will include an investigation of this approach.

The testbed uses the OpenSkies simulation software, developed at Cybernet Systems. The OpenSkies system is a state-of-the-art PC-based simulation software package designed from the ground up as a general-purpose simulation tool kit. OpenSkies supports such features as force feedback, accelerated 3D graphics, stereoscopic 3D graphics, and HLA-based multi-user networking.

It is an irrefutable fact that human beings currently possess adaptive capabilities that far exceed our machines. The best solution for UAV control is therefore an optimal combination of the human and technological component. Our goal is the creation of a system that provides a powerful tool to the human operator that performs the iterative, math-intensive details of command distribution and control, and frees him/her to view and plan UAV actions using an intuitive and geometric interface.

SWARM LEVEL COMMANDS

To control multiple UAVs, there are specific command requirements for the UAVMI system. Relevant issues include:

a) Where and how exactly does the UAV's autonomy interface with the higher level commands coming from the UAVMI? The solution is to create (or adopt an existing) command language for the individual UAVs.

b) Define a library of "swarm-level commands". These are commands that effect groups of UAVs. These commands will be generated within the UAVMI.

c) Address the issue of how "swarm-level commands" translate into commands to individual UAVs. For example: if the swarm-level command is for *group A* to attack enemy *group E*, then what exactly should a_1 (one UAV in *group A*) do? Attack e_1? Attack the closest element in *group E*? What if that target is already covered by a_3?

Ultimately, we want to understand the maximum number of UAVs a single operator, and a group of operators, can control. There is a concern that information overload will cause operators problems when they attempt to manage multiple UAVs. Fortunately, current work has shown that it is possible to manage multiple (dozens or more) UAVs simultaneously. Relevant issues include:

a) Command-group definition - The operator must have the ability to divide the UAV group under his/her command into subsets, each of which can be given separate commands. The operator should be able to perform this in real-time, which likely suggests a graphical selection process. Therefore we are developing a set of real-time command-group selection interfaces.

b) Other-group definition - The operator must have the ability to divide the set of all other known vehicles or installations into subsets. Similar to command-group definition, the user must be able to perform this in real-time. Furthermore, the operator must be able to characterize the group as a threat-type that may effect path planning. Therefore we are defining a set of real-time other-group selection and characterization interfaces.

c) Path generation - The operator will be able to command a UAV group to a location. One method being used is to allow the operator to, at a high level, tell the UAV group where to go. The software must then use these way-points in conjunction with information about threat location and terrain constraints to construct the fastest, safest path. Therefore we are defining the set of constraints incorporated into the system as well as the operator input.

d) Relative posture command - The operator must also have the capability to command a UAV group to a position relative to another entity/group. For example: command group A to attack enemy group E from azimuth of 270° and an elevation of 45° (i.e. from the east and from above).

A CUEING SYSTEM

To allow for multiple UAV control, a cueing interface is required, which includes a graphic representation of the operating theater presented to the user as well as audio rendering. Relevant issues include but are not limited to:

a) Graphical differentiation - The operator must be able to tell which forces are friendly, which are hostile, and which are the ones he/she is controlling. We are creating visual cues, such as highlighting and group color, that such differentiation.

b) Location cues - The operator must be able to perceive 3D location of the entities clearly, especially altitude. We are exploring such mechanisms as 3D stereoscopy, artificial shadows, and drop-to-terrain lines.

c) Audio cues - Sound can be used to warn the operator of proximity, velocity, and events such as missile launches, takeoffs, and landings. We are creating a set of audio cues for this system.

SELECTION METHODS FOR THE SWARM LEVEL TACTICAL INTERFACE

Selection is basically the mechanisms for specifying which UAVs are affected by commands and which are showing their information. The selection of units is a simple function of a swarm interface, necessary for specifying a subset of units to view information on and to command. We name two common approaches to selection. The first is "Single Selection", which is simply clicking on a unit directly or from a list of units. The second is "Region Selection", which is the action of specifying a geometric region (usually rectangular) using the mouse, and any units in that region are selected. Units selected will have a different appearance in the interface, usually done by either showing a translucent "box" around the unit, having a simple 2D graphic "float" above the unit, or changing the color of the unit. Below we list enhancements of these two basic selection methods.

- **Selection Tool Palette** – this is simply a palette, like in a drawing program, of different shapes that can be used for the region selection. This palette would include the basics such as rectangular, square, circular, and elliptic shapes, and add shapes such as polygonal, closed spline (a curvy polygon), and freeform. Since we are dealing with 3 dimensions, however, there also need to be 3D versions of those listed earlier. For the first four we listed, the 3D versions are box, cubic, spherical, and ellipsoid. Some other valuable 3D shapes are cylindrical, cone, and possibly torus (donut). Figure 1 shows an example selection palette, depicting square, circular, elliptic, spline, cubic, spherical, cylindrical, torus, and cone regions.

Figure 1. An Example Selection Palette

- **Grouping** – Numerous real-time strategy computer games have established an effective mechanism for splitting a large number of units into customizable groups. In general, a group is defined by selecting all those units to be in the group, and then the Ctrl key is pressed while selecting a number (0-9). This

"sets" the group to the number key pressed. A group can then be selected entirely by simply hitting the number key that corresponds to it. This can obviously be extended for functions keys, other keyboard keys, or even using other modifier keys (Alt, Shift) to specify other layers of keys. So we can have a group for the "1" key, another for the "Shift+1" key combination, another for "Alt+1" combination, and yet another for "Shift+Alt+1" combination. Units ***can*** be in more than one group, for instance if we have a group that contains a sensor drone, but also want a group for all sensor drones.

- **Standard GUI Conventions** – It only makes sense to use the conventions found in most software packages in use today. Some of these conventions, and their translation in the context of a swarm GUI are:

  - *Ctrl+Click* = Select a unit if not already selected, deselect it if it is selected. This also will add/remove a unit from the current group.
  - *Shift+Click* = Select all the units from the currently select unit to the unit shift-clicked on, the region selected could be some basic standard (probably a box), or maybe use the currently chosen region in the selection palette.
  - *Double-Click* = Select the unit and the group that the unit is currently in. If a unit is in more than one group, select the first (lowest number) group.
  - *Triple-Click* = Select all units in the area, or possibly all units entirely.

- **Fuzzy Selection** – Quite often, the need arises for the ability to quickly select a subset of some units to have them perform a command. For instance, a set of hostiles is detected and we want to take several, but not all of our units to respond to it. However, our units may be distributed close together, and in such a way that we can't simply select a group of them using a normal tool. What is needed is a "fuzzy selection". A fuzzy selection is a modifier of a normal region selection (rectangular or spherical for instance), but only a subset of the units that fall in the region are actually selected. The units actually selected can either be based on a percentage ("select a third of the units in this region") or by type ("select all sensor drones in this region") or even by other categories such as health, fuel, or anything ("select all the severely damaged units in this region"). Which types of fuzzy selection are most beneficial depends on the results of testing various options and seeing which are most used. Figure 2 shows a traditional, rectangular region selection and the resulting gap formed that may be tactically undesirable. Figure 3 shows the use of a 33% "fuzzy" selection, and the resultant formation that is much more tactically feasible.

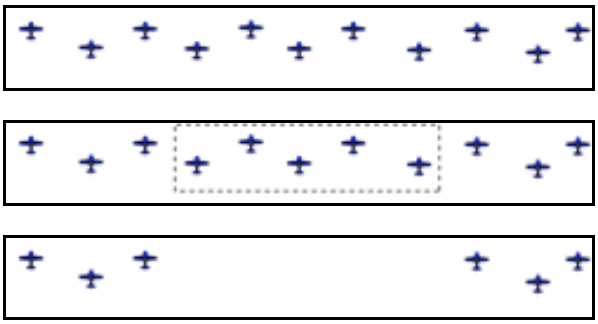

Figure 2. Traditional Selection Resulting in a Tactically Poor Formation

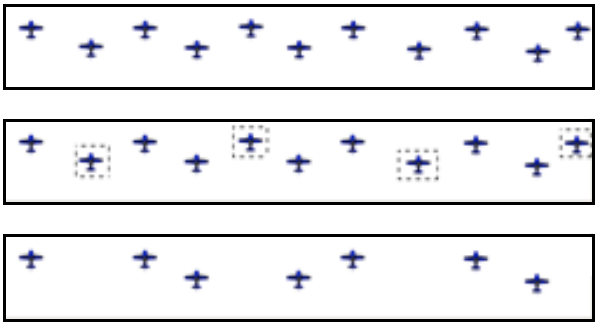

Figure 3. "Fuzzy" Selection of a Third of the Units in a Region, and the Resulting, Much More Tactically Friendly, Formation

VIEWPOINT NAVIGATION METHODS FOR THE SWARM LEVEL TACTICAL INTERFACE

Viewpoint navigation is simply the manner in which the operator "moves around" in the virtually represented space that is populated by the UAVs. A basic feature of a multiple UAV interface is the ability for the operator to move an "avatar" around. An avatar is simply the operator's presence in the simulation. An avatar is usually thought of as the camera in the world. Like a camera, there are many ways to position and orient an avatar. The primary methods that we are implementing for a multiple UAV interface are:

a) **Free Fly** – This mode allows the avatar to move around the world at will, using a simple set of controls. This mode was completed during the Phase I effort, and is included here for completeness. The avatar is not attached to any object, and not forced to look in any

direction. It is a useful mode when everything is going well and the operator wants to "take a look around". Typical control schemes are the use of the mouse to control the direction the camera is looking, and the keyboard to move the camera in the direction of the viewpoint, or reverse, or orthogonal to the viewpoint.

b) **First Person** – The first person mode puts the operator in the virtual seat of one of the UAVs. This mode was completed during the Phase I effort, and is included here for completeness. This mode is often coupled with the Wingman Command mode outlined in the Command section below, giving the operator more direct control of the UAV. It allows the operator to get a better feel for what a UAV can sense, and can be attached with a morphing of the GUI to give a more intimate feel for how a UAV is doing. A first person GUI would be similar to a flight simulator, in that all the normal flight instruments can be presented to the operator. Since the UAV is virtual, the operator can look around more freely than if there was simply a camera attached to the UAV, allowing the operator to look straight down, backward or whatever direction desired. Control in the mode is usually limited to orientation of the camera, through the use of the mouse or keyboard, although positional control can be obtained when direct control of a UAV is active.

c) **Third Person** – Third person allows the operator to "orbit" a single UAV, with similar advantages, but allowing for more "body sense" of the UAV by allowing the operator to see what the UAV is doing. This mode was completed during the Phase I effort, and is included here for completeness. The camera is attached to the UAV but is outside of the UAV, at a distance specified by the operator. The avatar can move around the UAV, but the camera is always pointed towards it. Navigating tight situations is a good use for this mode, as is the communication of vehicular damage. Third person control schemes usually have the mouse movement control the rotation around the UAV, and the mouse wheel or keyboard to zoom in and out. Through the use of standard "trackball" control algorithms, this movement is extremely intuitive.

d) **Bird's Eye** – The bird's eye view is similar to radar in that the camera is high above the ground, looking straight down, creating a type of 2-D representation. This allows for "big picture" information to be communicated quickly, since humans are very adept at dealing with 2D representations of information. The avatar always points down, but can zoom in and out, and move around laterally. Control of the bird's eye view is fairly simple – arrow keys to control movement in the plane or the operator can "bump" the mouse pointer against the sides of the screen for similar control, and then either another two keys or the mouse wheel to zoom in and out.

e) **Center of Mass** -- This is similar to the Third Person mode, but instead of being attached to a single UAV, the avatar orbits the center of mass of a group of UAVs. This allows the operator to more easily deal with group commands and group information. The changing nature of the UAVs has minimal impact on the camera, yet the camera will move with the group as a whole, making it a bit easier to navigate in some circumstances. Unlike the bird's eye view, this mode allows the operator to interact with the UAVs in all three dimensions. The controls for the Center of Mass mode operates very similarly to the third person mode, except the center of mass is focused on instead. A prototype of this viewpoint navigation method will be complete by the end of the Phase I effort.

f) **Tactical Multi-View** -- This mode is useful for viewing all of the UAVs on the screen, without the need for exact distances between them. One instantiation of this technique is to display a local area map, with multiple "windows" showing a bird's eye view of each UAV. See Figure 4 for an example of this effect. This technique allows for both an intimate view of what each UAV is doing, while also giving a higher-level view for force deployment, and other, more abstract needs.

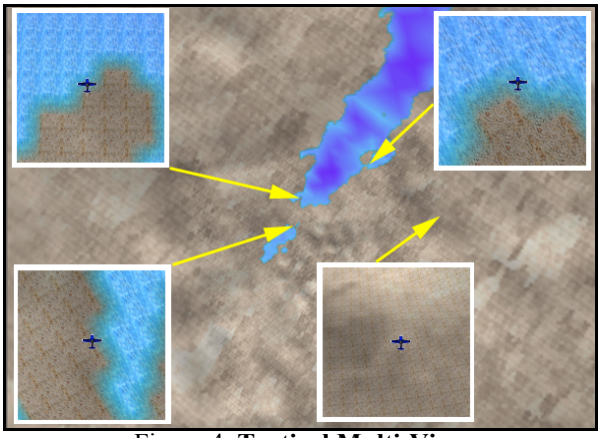

Figure 4. **Tactical Multi-View**

INFORMATION DISPLAY METHODS FOR THE SWARM LEVEL TACTICAL INTERFACE

Unique approaches must be taken to display the information about UAVs in such a manner that won't overload the operator, yet allows important individual and swarm information to be noticed and dealt with in a timely manner. The display of information, including unit information, group information, environmental information, and current mission information, amongst others, is a complex topic. We address some of the more basic concepts, with some concrete examples in this section.

The primary focus of the interface when it comes to informational display is to maximize the amount of information communicated while minimizing the attention needs from the operator. Showing all the relevant statistics for all units will quickly cause information overload. The trick is to show information only when it is needed, so there is no point in always showing the amount of fuel in all units. Instead, fuel status is only needed when it becomes low, in general circumstances. However, to compound the "trickiness" of the situation, exactly what is important to display is context dependant. Continuing the fuel example, while normally we don't want to know fuel levels unless they get below a certain level (the amount needed to return to home base, for instance), some contexts may require more information. If we command a UAV to go to a certain location, we then want to know if there is enough fuel to get there (and back). Furthermore, important information seems to avoid coming in a nice steady stream. Instead, it often "clumps". A group of units suddenly discover a convoy of enemies, while the convoy also discovers the units, sending a wad of information the operator's way.

We address these issues in the following general techniques.

- **Per Unit Graphics** – Numerous techniques have been developed over the years of presenting the most important information about an entity with small graphical "widgets". In the case of a UAV, this will probably be a line graphic for an abstraction of its "health", and perhaps fuel and armament. Small icons can be used to show state information ("Defensive Stance", or "Attacking" for instance). Figure 5 shows an example of possible icons.

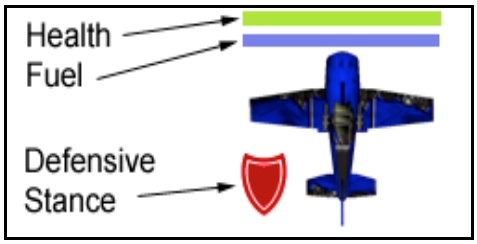

Figure 5. Example Graphic Icons for a Unit

- **Message Console** – A message console can be displayed at the bottom of the screen. This console is simply an output for text message from any entity in the interface. It can contain messages from UAVs ("Fuel Low!" or "Under Attack!") or environmental messages ("Thunderstorm predicted to hit at 15:30"), for instance.
- **Event Queue** – To handle the issue of information overload because of multiple simultaneous events, we present the idea of an event queue. What happens is that an event occurs, but instead of a popup or some other intrusive action occurring that could interrupt what the operator is currently doing, an alert icon is flashed briefly along with an audible, then the actual event is placed on a graphical queue of events on the side of the screen. The operator can then handle the events in an orderly fashion, either by picking individual events out of the queue, or through the use of a "Next Event" command. Events in the queue can have priorities, which can either affect the ordering of the events in the queue, or simply cause the icons in the queue to have a different color based on the priority.
- **Command Prediction** – As mentioned above, the importance of information is often context based. One situation where this is true is in giving commands to UAVs. An operator not only wants to know the current status of the UAV, but whether the UAV will be capable of carrying out a given command. A command to attack is useless if a unit has no armaments. A simple method for this is a "rejection" tool, whereby a UAV rejects a given command in some manner clearly noticeable by the operator. An audible buzz and a red flash around the unit is an example.
- **Access to Detailed Information** – While we want to minimize information displayed, the operator should always be able to get to the information needed, through the use of special side-screens or popups. Selecting a single UAV, for instance, should bring up most of

the statistics about the UAV in a small side window. Another method is too have a small "balloon" popup when the mouse pointer is held over a unit, similar to "balloon help" that exists in most operating systems. Detailed information will also include special sensor information that will either adjust the individual UAV display, the swarm UAV display, or the actual map interface that the UAVs interact with.

COMMAND METHODS FOR THE SWARM LEVEL TACTICAL INTERFACE

Quick and rapid techniques must be developed to allow operators to give orders to a swarm of UAVs, while minimizing the amount of functionality lost when compared to a single operator controlling a single UAV. The most important piece of the interface is the one that allows the operator to give commands to the UAVs. Complex sets of commands must be easily specified and delivered. The previous sections really deal with helping the operator determine what commands to give, how current commands are unfolding, and specifying which UAVs receive which commands. In this section we identify not only how to give commands, but some interesting types of commands to give.

- **Travel** – The most basic command to give to a UAV is "go to there". This is simply done in our proposed interface by selecting a UAV, and specifying the location it is to go to. The same technique can be used with multiple UAVs (groups). One nontrivial portion of this is exactly how the destination is specified. Two dimensions is simple, but the addition of the third makes it much more complex. For a solution, we turn to the 3D RTS game "Homeworld" (see Homeworld, 1999). In this game, to move a unit, the unit is selected, and then the "move" command is given. This brings up a "movement disk" that is gradated with the degrees of the compass. Moving the mouse around moves within this disk, and the path specified is shown as a line from the unit to the current position of the mouse in the disk. The operator can hold the shift key at any time, and this will allow the operator to modify the altitude up and down. The operator can intersperse changes in the 2D disk and in elevation. Once the destination is correct, clicking the left mouse button sends the unit there.

- **Waypoints** – Beyond the first simple movement command, another idea borrowed from the computer gaming world is waypoints – the ability to specify multiple destination points for a unit, which will be followed in turn. New waypoints can be added through a special "Add Movement" command.

- **Formations** – Often, it is advantageous for units to travel in a specific formation. To accomplish this in the swarm interface, we will create a simple "formation palette", that allows the operator to select different shapes for the UAVs to fly in. Figure 6 shows an example palette, although it only displays two-dimensional formations. Other formations will need to be included that capture the possibility of using all three dimensions.

Figure 6. Formation Palette

- **Distributions** – Once at a location, a group of UAVs may form a different, dynamic pattern to fulfill a certain duty. For instance, a group of sensor drones may travel to an area in a wedge formation, but once there, it may form a circle around a specific area to form a perimeter to prevent something from passing out of an area undetected. We call this "distributions". Similar to formations, a palette of formations will be available to the operator. These distributions will tend to be a bit more complex, and as such, the operator can combine multiple distributions over time. For instance, continuing the sensor drone example, the circle may start rather broad (50 km radius), but if the mission changes to fine the object of interest, the circle may start to "close". This could be a combination of a "circle" distribution with a "inwards" distribution. One important component of a distribution is the "holding pattern". UAVs may be commanded to simply hover, or tightly turn, in place, or the distribution may be cyclic.

- **UAV-Specific Commands** – Many of the commands given to a UAV will be specific to the type of UAV it is. Different UAVs have different munitions, sensors, and general capabilities. Clicking on a UAV will bring up a palette of commands corresponding to that UAV.

- **Stance/Attitude** – Often, things will happen to a UAV while the operator is not directly observing the UAV. In this situation, the UAV may have automated responses. We call such automation its "Stance". The stance of a UAV is indicated by a small icon near the UAV on screen, as well as in any informational display of the UAV. Some stances are complicated and require other information to be specified, such as a location to flee to, or how much munitions to expend when attacking. Some possible stances are:
    - ***Hostile*** – if something comes near the UAV, it attacks.
    - ***Guarding*** – if something attacks the object the UAV guards, the UAV attacks.
    - ***Defensive*** – if something attacks the UAV, it responds in kind.
    - ***Observing*** – if something attacks the UAV, the UAV moves away.
    - ***Spy*** -- the UAV tries to remain undetected, but if it is, it moves with all haste back to a location.
- **Command Queuing** – Commands can be queued for a UAV or group of UAVs. The queue is displayed when the unit is selected, and can be modified. So a unit can be told to fly to a certain location, and then release its bombs on a target, and then fly to a different location and enter into a sensor-sweep mode. This allows the operator to build up long commands from a smaller set of "commandlets".
- **Wingman Mode** – One mode possible for the command and control of a UAV is the so-called "Wingman Mode", where the operator takes almost direct control of a UAV, and treats the other UAVs as wingmen. In this mode, the display is replaced with more detailed instrument information to allow the operator to control the UAV directly. Furthermore, commands in this mode are much more simplistic and terse, since the operator's attention will be more consumed by the act of controlling the UAV. Sample commands in this mode are:
    - **"Follow Me"** – all the UAVs go into a predefined formation and follow the operator's UAV as the lead.
    - **"Attack My Target" –** the operator can specify a target and send its wingman UAVs to attack it.
    - **"Bomb My Target" –** Similar to "Attack my target" except the target is ground-based.
    - **"Defend Me" –** Prevent hostiles from damaging the controller's UAV.

CONCLUSIONS

We have described innovative interface for operators to control multiple UAVs in a combat situation. We have detailed selection methods, navigation methods, display methods, and command methods. While still a work in progress, many of the methods here have been implemented in our OpenSkies system, and they lay the foundation for full control of swarms of UAVs in a combat environment.

REFERENCES

Nelson Minar, Roger Burkhart, Chris Langton, Manor Askenazi, 1996. *The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations (1996).* Retrieved June 27, 2003 from http://www.santafe.edu/projects/swarm/

Colonel Bruce Stark, USAF, 2003. Dynamic BMC4I – the Central Nervous System of the Command and Control Constellation (C2C). *UCAV 2003, March 25-26 2003, Alexandria, VA.*

Dr. Michael S. Francis, Colonel, USAF, 2003. UCAV – Catalyst for Network Centric Operations. Teamwork in Cyberspace: Using TEAMCORE to Make Agents Team-Ready. *UCAV 2003, March 25-26 2003, Alexandria, VA.*

Tambe, Shen, Mataric, Pynadath, Goldberg, Modi, Qiu, and Salem, 1999. *1999 AAAI Spring Symposium Series, March 22-24 1999, Palo Alto, California.*

Larimer, 1997. Thesis: Building an Object Model of a Legacy Simulation." Retrieved June 27, 2003, from http://diana.or.nps.navy.mil/~ahbuss/StudentTheses/LarimerThesis.pdf

Homeworld, 1999. Published by Sierra Games. Retrived June 27, 2003, from http://www.sierra.com/product.do?gamePlatformId=110