The Rescue Manager App: uFldRescueMgr

May 2024

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Overview

The uFldRescueMgr app, a.k.a. "rescue manager", was developed to support the MIT 2.680 Autonomous Rescue Lab. This lab is a multi-week lab in both simulation and in the water with four autonomus robots. The rescue manager is the central course-provided software to enable this lab sequence.

The uFldRescueMgr app is the shoreside arbiter for the Autonomous Rescue Lab sequence and competition. It holds the ground truth information of swimmer locations and rescue status. Ground truth locations are read in from a swim file, o may be added dynamically on the shoreside by an

operator. The status of the swimmers (rescued/unrescued, found/unfound) are maintained by the rescue manager but change status based on locations of the rescue vehicles or scout vehicles. The rescue manager manages the games state in terms of team tallies and declaration of a winning team. The rescue manager is available to participants when running test simulations, but will be run on a shoreside computer by a competition manager (class TA) during an competition, simulated or in-water.

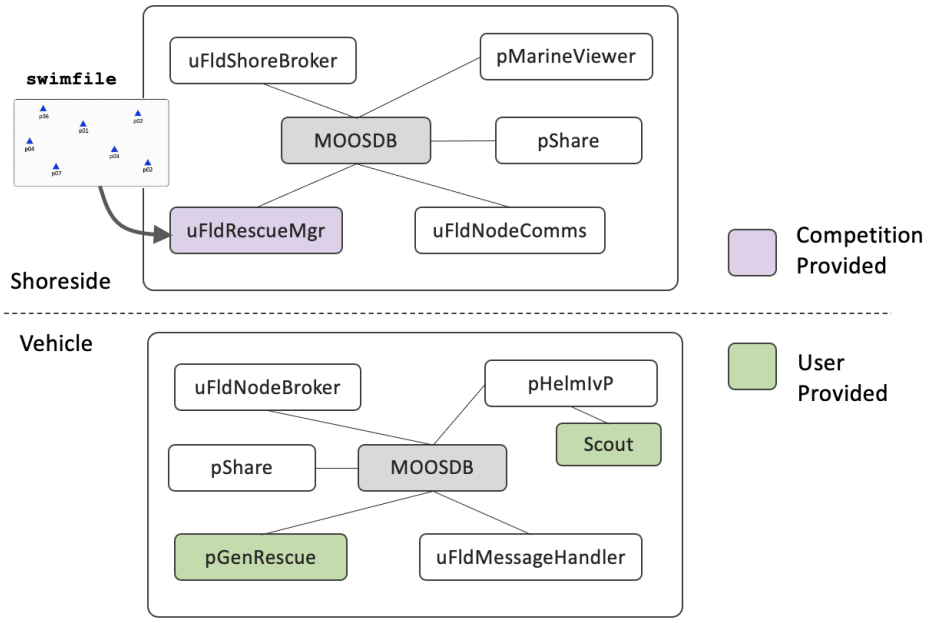

Figure 1: **The uFldRescueMgr**: is run in the shoreside community. It loads the ground truth of swimmers and locations from an swim file, and will convey this information to the vehicles when or if they are close enough to the swimmer. The resecue mananger will also grant or denie rescue requests from vehicles, and keep track of the rescue state for each swimmer, and tally of total swimmers rescued by each vehicle. The `pGenRescue` app is typically provided by competitors as the policy for visiting swimmers and attempting rescues.

2 Configuring a Mission to Use the Rescue Manager

To use the rescue manager, `uFldRescueMgr`, three steps are required:

- The `uFldRescueMgr` app must be added to the list of apps running on the shoreside.
- The key variables needed by the rescue manager from the vehicles, and the variables published out to the vehicles, must be configured for sharing.
- The rescue manager configuration block must be added to the shoreside mission file with desired parameter values if they differ from the default values. The name of the *swim file* must also be declared in this configuration block.

These steps are described below. However, if using the rescue baseline mission folder as part of MIT 2.680, many of these steps are already in place in that folder. Users are welcome to change things

during simulation testing and development, but the shoreside settings will largely be controlled by a competition administrator in a head-head competition and the settings will not be changeable by the participants fielding vehicles.

2.1 Running the Rescue Manager

The rescue manager is run on the shoreside MOOS community. If using the rescue baseline mission folder, this rescue manager is already present in the shoreside mission file (`meta_shoreside.moos`). It is present in the Antler configuration block with a line:

```
Run = uFldRescueMgr  @ NewConsole = false
```

In addition, the rescue manager will have a configuration block in the same mission file. Publications of the rescue manager sent to the vehicles, in the form of MOOS variables over pShare, must all be configured in the uFldShoreBroker configuration block in the shoreside mission file. More is said about these latter two steps in the following sections below. Altogether, these three steps are all that is needed to use the rescue manager. Just to reiterate, if using the rescue baseline mission, these three steps are already implemented in the shoreside mission file. Their description here is solely for reference and in case any of the parameters may need to be changed in the future.

2.2 Required MOOS Variable Shares to Vehicles

Using uFldRescueMgr requires certain information flowing between the shoreside and vehicles communities as shown in Figure 1. The sharing is done by pShare, but the pShare configuration is handled dynamically using the uFldNodeBroker and uFldShoreBroker applications, we discuss here the necessary configuration entries for these two applications. From the vehicle to the shoreside, three key variables need to be shared. The below lines should appear in the uFldNodeBroker configuration block on all vehicles.

```
// In uFldNodeBroker configuration
// Shares from Vehicle to Shoreside specific to uFldRescueMgr

bridge =  src=NODE_REPORT_LOCAL, alias=NODE_REPORT   // likely have already
bridge =  src=RESCUE_REQUEST
bridge =  src=SCOUT_REQUEST
```

The first line, for sharing the NODE_REPORT, would likely already be present due to their use in other applications. The rescue manager needs to know vehicle locations to enable it to determine whether rescue attempts are granted. The latter two variables are generated by vehicles participating in the competition. They could originate in a user MOOS app, or simply in a timer script on the vehicle.

The below four lines should appear in the uFldShoreBroker configuration block in the shoreside MOOS community (along with likely several other lines). See uFldShoreBroker documentation for a discussion on the syntax.

```
// Shares from Shoreside to Vehicle - in uFldShoreBroker config
```

```
bridge =  src=RESCUED_SWIMMER
bridge =  src=SCOUTED_SWIMMER_$V,     alias=SCOUTED_SWIMMER
bridge =  src=SWIMMER_ALERT_$V,       alias=SWIMMER_ALERT
bridge =  src=NODE_COLOR_CHANGE_$V,   alias=NODE_COLOR_CHANGE_REQUEST
```

The lines that have a source component ending `$V` configure MOOS variables intended to be shared to a particular vehicle. For example, a post to `SCOUTED_SWIMMER_ABE` by `uFldRescueMgr` on the shoreside will be shared only to vehicle abe, and it will arrive on abe as the variable `SCOUTED_SWIMMER`. On the other hand, a post to `RESCUED_SWIMMER` will be shared to all vehicles. This is by design since a swimmer rescue is meant to be known by everyone when it happens, whereas the information about a previously unknown swimmer is meant to be shared only to the scout vehicle that came across the new swimmer. For more information on bridging, see the documentation for `uFldShoreBroker`. (Type `uFldShoreBroker -w` on the command line.)

2.3 Configuring the Rescue Manger

The rescue manager has a number of configuration parameters dictating both the performance of the app as well as preferences for visual rendering of state. The set of configuration options is listed thouroughly in Section 5, with default values. It should be noted that the configuration for this app in the rescue baseline mission may occasionally have different parameter values.

A key configuration parameter for `uFldRescueMgr` is the specification of the *swim file*. This file contains the ground truth of swimmers and their locations, and is typically found in the mission folder where the shoreside MOOS community is launched. This is described next.

3 Swimmers and Swim Files

The rescue manager reasons over *swimmers*. A *swim file* is a plain text file specifying a collection of swimmers.

3.1 Swimmers

A *swimmer* has the following characteristics:

- The `(x,y)` location.
- The swimmer `name`.
- The swimmer `type`.

The swimmer location will reside in prescribed search region given by a convex polygon. The name is unique to each swimmer. The type is either `registered` or `unregistered`. In a swim file, a particular swimmer entry will be described in a single line, e.g.,

```
swimmer = type=reg, name=p03, x=34, y=98
```

During the course of a competition, a swimmer will also have the following state:

- The swimmer `state`: either rescued or unrescued.
- The vehicle `savior`: a vehicle name that rescued the swimmer, or an empty string if unrescued.
- The UTC `time` of rescue or -1 if unrescued.
- The swimmer ID: an alias for the swimmer name.
- The set of scouted vehicles: Vehicles that know about the swimmer, in the case of unregistered swimmers only.

At the outset of the mission, the ID and position of registered swimmers are sent to all vehicles with `SWIMMER_ALERT` message:

```
SWIMMER_ALERT_ABE = id=38, x=45,y=98
```

Alerts are sent individually to each vehicle.

3.2 Registered vs. Unregistered Swimmers

The optional use of unregistered swimmers supports a mission variation where scout vehicles work in collaboration with rescue vehicles. It is this feature of `uFldRescueMgr` that enables the "collaborative" autonomy aspect of this rescue lab. Without unregistered swimmers, there would be no use for a scout vehicle.

Registered and unregistered swimmers differ primarily in that the former are broadcast to all vehicles at launch time and continuously throughout a mission. Unregistered swimers are known to the rescue manager at mission start time and only become known to a vehicle if the a *scout vehicle* discovers the unregistered swimmer. An unregistered swimmer may be known to no vehicles, one vehicle, or multiple vehicles.

At the outset, registered swimmers are rendered in a single neutral colored triangle. In the rescue baseline mission this is set to lime. Unregistered swimmers are rendered initially as two half-circles, with both halfs initially transparent. Two transparent halves means the unregisered swimmer is not known to either team. If half the circle is colored, say yellow, this indicates the yellow team knows about the swimmer. If both halfs are colored, say yellow and red, then both the red team and yellow team know about the swimmer, as in Figure 2 below.

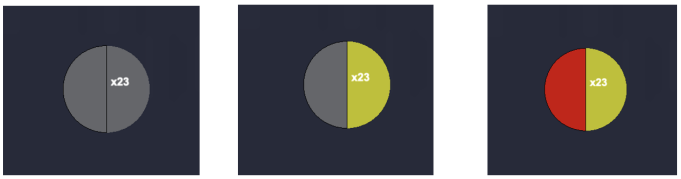

Figure 2: **Unregistered Swimmers**: An unregistered swimmer known to neither team is rendered with two colorless half circles (left). An unregistered swimmer known only to the yellow team (center). An unregistered swimmer known to both the red and yellow teams (right).

The renderings are just for the sake of awareness for the user looking at the GUI. Vehicles on one team are not aware about what knowledge the other vehicles on the other team have.

If a vehicle from say the yellow team later rescues this swimmer, the swimmer will be rendered with a yellow triangle. See Figure 3.

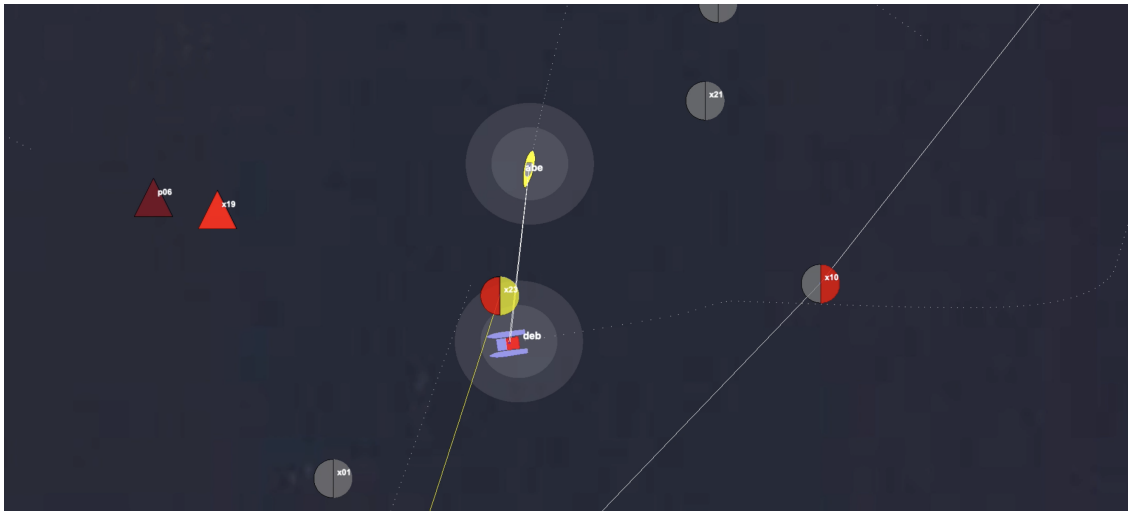

Figure 3: **Scout and Tell**: In this game snippet, first the yellow scout vehicle discovers the swimmer x23. After discovery, the scout vehicle immediately sends a message to the yellow rescue vehicle with the location of the newly found swimmer. A short time later, the red scout vehicle discovers the same swimmer and also informs its rescue vehicle teammate. At this point the unregisted swimmer is rendered as half red, half yellow. Finally the yellow rescue vehicle swoops in and rescues this unregistered swimmer after which it is rendered simply as a yellow triangle.

3.3 Swim Files

Swim files are used for seeding the rescue mission with a number of swimmers and their locations. They are plain text files usually generated from a command line utility called gen_swimmers discussed in a following seciton. The utility is provided with a polygon region from which to choose random locations, and a constraint on the minimum distance between swimmers.

Below is an example file. Note the first line is a comment, showing the exact command line argument used in creating the file. The second line is also a comment showing the actual minimum distance between any two swimmers. In this example the requested min distance was 10 meters, and the resulting file satisfies this constraint. the third line is the polygon region from which the random swimmer locations were selected.

```
// gen_swimmers --pav60 --swimmers=5 --sep=10
// Lowest dist between swimmers: 11.40
poly = pts={60,10:-30.3602,-32.8374:-4.6578,-87.0535:85.7024,-44.2161}
swimmer = name=p01, x=57, y=-1
swimmer = name=p02, x=54, y=-12
swimmer = name=p03, x=-12, y=-33
swimmer = name=p04, x=-9, y=-66
swimmer = name=p05, x=-11, y=-49
```

The swim file is read as a coniguration parmeter in the configuration block of uFldRescueMgr. The

polygon is read in by uFldRescueMgr as well as the swimmers. The polygon region is also broadcast to all vehicles. Vehicles may use this polygon region as in input parameter for the OpRegion helm behavior.

In the example above, each swimmer is assumed to be a registered swimmer since the type was not specified. For swim files containing unregistered swimmers, like the example below, each swimmer line will be explicit in declaring the swimmer type.

```
// gen_swimmers --pav60 --swimmers=5 --unreg=4
// Lowest dist between swimmers: 15.00
poly = pts={60,10:-30.3602,-32.8374:-4.6578,-87.0535:85.7024,-44.2161}
swimmer = type=reg, name=p01, x=66, y=-12
swimmer = type=reg, name=p02, x=53, y=-49
swimmer = type=reg, name=p03, x=12, y=-37
swimmer = type=reg, name=p04, x=35, y=-22
swimmer = type=reg, name=p05, x=59, y=-26
swimmer = type=unreg, name=x01, x=30, y=-53
swimmer = type=unreg, name=x02, x=2, y=-82
swimmer = type=unreg, name=x03, x=44, y=-37
swimmer = type=unreg, name=x04, x=69, y=-42
```

Note that registered swimmers are names `p01`, `p02` up to say `p45` for 45 registered swimmers. Unregistered swimmers use the labelling `x01`, `x02` and so on. The swimmer *name* is only showed on the GUI. Vehicles receive a scrambled ID number so the vehicles are not able to infer the total number of swimmers of any type.

Note: The swim file is logged along with the shoreside alog file and shoreside mission file, to be clear later if there is any doubt about the actual swim file used in a mission. This is ensured by the rescue manager by posting

```
PLOGGER_CMD = COPY_FILE_REQUEST=swim_file.txt
```

The PLOGGER_CMD is read by pLogger and is useful feature baked into logger that comes with MOOS. The file `swim_file._txt` will be found in the log folder.

3.4 Generating Swim Files

Swim files are created using the gen_swimmers command-line utility. This utility generates output to the terminal. A swim file can be created by re-directing the output to a file. For example:

```
$ gen_swimmers --pav60 --swimmers=15 --unreg=5 --buf=10 > swim_file.txt
```

The result is just a plain text file and can simply be edited. But the command-line utility ensures (a) all swimmer locations are within the given polygon rescue region, and (b) the minimum distance between swimmers is respected if it is specified. The utility is passed a convex polygon command line argument describing the region from which to randomly pick swimmer locations. For example:

```
$ gen_swimmers --poly=60,10:-30.4,-32.84:-4.7,-87.1:85.7,-44.2 --swimmers=10
```

Or the options `--pav60` or `--pav90` can be used as shortcuts for the two common operation areas at the MIT Sailing Pavilion and used in the rescue baseline mission.

To see more complete documentation of this utility, run:

```
$ gen_swimmers --web
```

The `gen_swimmers` utility is in `moos-ivp-2680` tree.

3.5 Viewing a Swim File

The `swimview` command-line utility enables a user to visually preview the swim file with a visual background of the rescue region polylgon. Without this utility, the only other option for visualizing the layout of swimmers for a give swim file would be to launch a simulation utilizing this swim file. This utility allows the visualization without all the simulation, and the user can toggle between several choices for quick comparison.

Launching from the command-line:

```
$ swimview swim_file1.txt swim_file2.txt swim_file3.txt ...
```

The viewer will allow for toggling through different swim files with the `[` and `]` keys. Figure 4 conveys the idea.

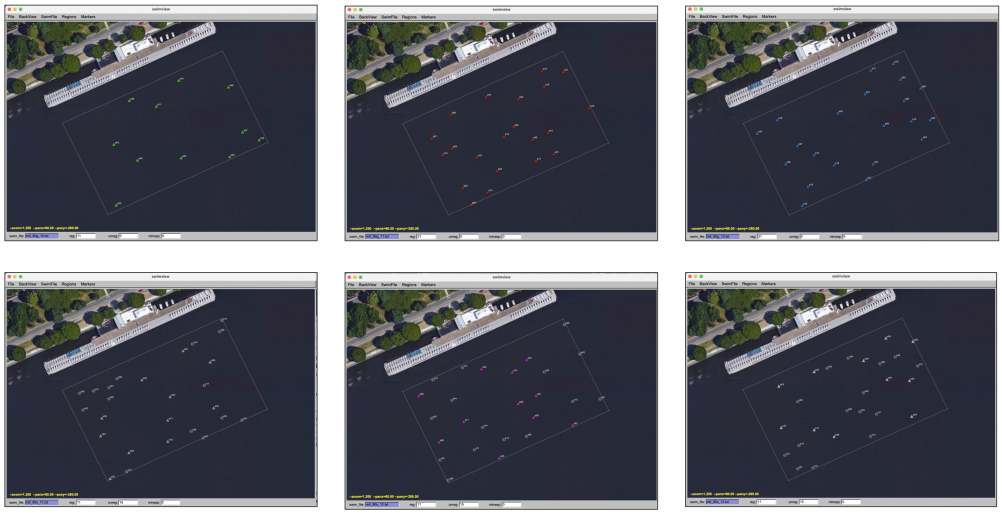

Figure 4: **The swimview Utility**: Multiple swimfiles can be loaded for quick visualization and comparison between swimmer layouts.

4 Operation of the Rescue Manager

A mission involving the rescue manager is mission constituting an adversarial head-to-head competition. The rescue manager needs to (a) know the ground truth in terms of swimmer and vehicle locations, (b) handle interactions with the vehicle including rescue attempts and scout/sensor requests, and (c) manage and convey game state to both the set of vehicles as well as in the shoreside operational display.

4.1 Rescuing Swimmers

The core interaction with a vehicle comes when a vehicle attempts to rescue a swimmer. In short, a vehicle will score a successful rescue when (a) it is sufficiently close to an un-rescued swimmer and (b) the vehicle generates a rescue request message received on the shoreside by the rescue manager.

A rescue request coming from a vehicle will look like:

```
RESCUE_REQUEST = vname=abe
```

The vehicle name is all the information needed by the rescue manager. If there is a yet-to-be rescued swimmer sufficiently closs to the named vehicle, the rescue manager will respond by posting a message like the one below. This message will be sent to *all* vehicles.

```
RESCUED_SWIMMER = id=07,finder=abe (From shoreside to all vehicles)
```

The reason it is sent to all vehicles is to allow other vehicles to adjust their plans accordingly, presumably removing that swimmer from any tour of waypoints currently being traversed. The ID component makes clear which swimmer has been rescued. All swimmers announced to the vehicles at the outset have the same ID values for the same swimmers. The `finder` field is also conveyed to a vehicle attempting a rescue can be sure that their rescue attempt was successful and not perhaps another nearby vehicle.

4.2 Rescue Criteria

A rescue request is granted if (a) a rescue request is received from a vehicle as discussed above, (b) the swimmer has not already been rescued, (c) the rescue vehicle is sufficiently close to the swimmer. The rescue range (meters) is determined by two configuration parameters:

- `rescue_rng_min`: Range within which a rescue request is has a probability of `rescue_rng_pd`.
- `rescue_rng_max`: Range beyond which a rescue request has a probability of 0.
- `rescue_rng_pd`: Maximum rescue probability for a rescue request.

The probability, p, of a given request being granted is:

$$p = \begin{cases} p_d & r < r_{min} \\ \frac{r_{max} - r}{r_{max} - r_{min}} \cdot p_d & r_{min} \leq r \leq r_{max} \\ 0 & otherwise \end{cases} \quad (1)$$

The two ranges can be rendered with two conenctric circles, as depicted in Figure 5. The posting of these circle can be disabled by setting `show_rescue_range=false`. The default value is true. Furthermore, the transparency of the circles can be altered by setting `rescue_range_transparency=N` where N is in the range `[0,1]`, where 1 is completely opaque and 0 is completely transparent. The default value is `0.1` .

4.3 Rescuing Swimmers Visuals

When a swimmer has been rescued, this will be apparent in couple ways visually through `pMarineViewer`. First, the appcasting output of the `uFldRescueMgr` app will change in three ways. (1) the total number of swimmers rescued for each vehicle is displayed, (2) the status of each swimmer is shown in an table with the name of the rescuing vehicle if it has been rescued, and (3) the recent-event portion at the bottom of the appcasting window will show/confirm the most recently rescued swimmer. See Section 7 for more on the appcasting output of the rescue manager.

The other apparent change is more readily visible: the swimmer will change color. It should take on the color of the vehicle that was granted the rescue. For this reason, the designer of the mission needs to take a little care to choose a color for unrescued swimmers that is different for any colors that will be used by vehicles. In the `rescue_baseline` mission, the unrescued color is set to `"lime"`.

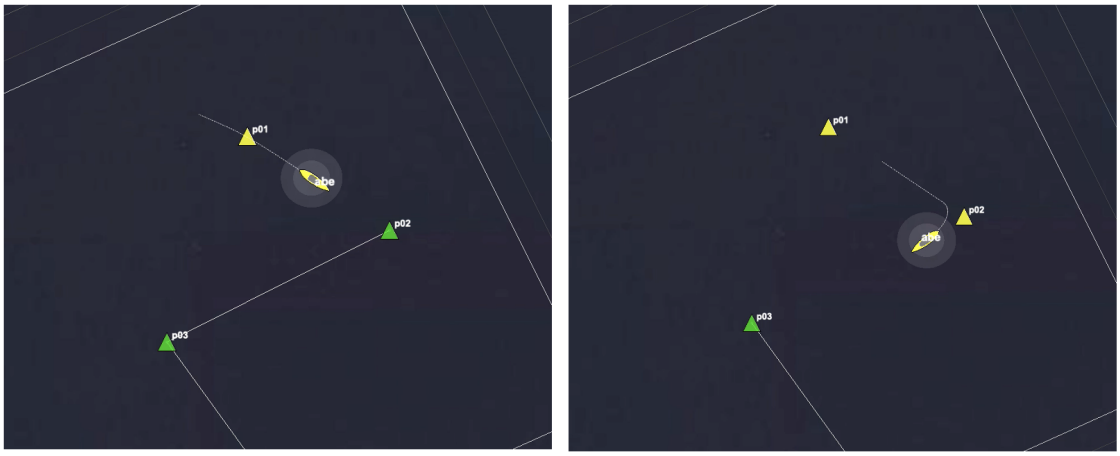

Figure 5: A the yellow vehicle rescues swimmers, the swimmer marker changes from the neutral color (lime) to the color of the vehicle that achieved the rescue.

As discussed further in Section 4.5, the color of the marker may take on a bright or dim color. If bright, then the marker represents a differential over the scores made by a competitor. For example if a red team has made three rescues, and the yellow team five, then there will be shown three dimly colored red markers, three dimly lit yellow markers, and two bright yellow markers. And where there is a tie, all markers will have a dim color.

4.4 Finding (Scouting) Unregistered Swimmers

The rescue manager knows about two types of vehicle roles, *rescue* and *scout*. The goal of a scout vehicle is to look for, discover and report the existence and location of unregistered swimmers. An unregistered swimmer is initially unknown to both teams, and eventually may be known either or both teams.

A scout vehicle may generate a scout request in the form of a message similar to:

```
SCOUT_REQUEST = vname=cal, tmate=abe
```

If the scout request is "fresh", e.g., received within the last five seconds, the rescue manager on each iteration will examine the vehicle location relative to each swimmer. Each examination will essentially involve a dice-roll. A successful dice-roll will generate a message back to the scouting vehicle of the form:

```
SCOUTED_SWIMMER_CAL = id=34, x=90, y=-2
```

The `_CAL` portion of the above posted MOOS variable matches the `vname` component of the scout request in the example above. It is sent only to this vehicle. Presumably the scout vehicle is then free to send a message to its rescue vehicle teammate about the presence of a new swimmer. The success or failure of the dice roll depends on the three parameters described in Section 4.2.

If a rescue vehicle comes across an unregistered swimmer that has not yet been rescued, it can be rescued by the rescue vehicle. It does not need to have been informed by its scout vehicle teammate.

4.5 Tracking Game State, and Game State Notifications

The rescue manager also keeps score. It keeps a running tally of total swimmers rescued by each vehicle. Every time there is a leader change, the rescue manager posts:

```
UFRM_LEADER = vname
```

The `vname` is the name of the rescue vehicle, or the word `"tie"` if the game is tied. The first such posting during a competition will be for the first vehicle that makes a rescue. There will be no initial posting `UFRM_LEADER=tie` at the outset.

As discussed previously in Section 4.3, the color of the marker for a rescued swimmer may take on a bright or dim color. If bright, then the marker represents a differential over the scores made by a competitor. For example, as in Figure 6 below, if a yellow team has made three rescues, and the red team two, then there will be shown two dimly colored red markers, two dimly lit yellow markers, and one bright yellow marker. And Where there is a tie, all markers will have a dim color.

Figure 6: As the vehicles rescue swimmers, the swim marker turns the color of the vehicle making the rescue. The team that has rescued more swimmers will render the markers representing the differential with brighter colors. In this example, The yellow team has scored three, and the red team two. There for there is one bright yellow marker shown.

The rescue manager can be configured to post one or more MOOS flags at the moment the leader changes value, set in the `leader_flag` configuration parameters. For example:

```
leader_flag = SAY_MOOS = $LEADER is the current leader
```

With the above, if the `iSay` app is also running on the shoreside, the mission could be configured to make an audible declaration of a leader change.
The game state, with further details of totals for each team, can be gleaned from the appcasting output of the rescue manager app, as discussed in Section 7.

4.6 End of Game Criteria

Winning and Finishing

There are two end-game notions: when the game has been *won*, and when the game is *finished*. A game is won when the winning vehicle team has been cemented. A game is finished when all known swimmers have been rescued. The reason for continuing after a game has been won is to allow the winning team to establish the dominance of their win. There are two reasons for this: (a) in certain circumstances an overall competition may be defined as the cumulative score across individual competitions, (b) one way of measuring the effect of an algorithm improvement may be to measure the improved dominance over a baseline opponent. It may be convenient normalize the winning score in the range of the total number of swimmers.

The competition can be configured to finish when a winner has been cemented with the configuration parameter:

```
finish_upon_win = true
```

The default is false.

Winning in the Rescue-Rescue Mission

A game is won when one team's rescue vehicle has rescued the majority of swimmers known to either team at that time. In the case of the one-on-one rescue-rescue variation, with no unregistered swimmers, this is pretty unambiguous. For these missions a swim file with an odd number of swimmers is always preferred.

Winning in the Rescue-Scout2 Mission

In the two-on-two rescue-scout2, a.k.a. "rs2" mission, further clarification is warranted: the rescue manager knows, at any given point in time, the count of registered, scouted unregistered vehicles, and unscouted registered vehicles. The sum of the first two groups is the total of presently "known" vehicles. As the scout vehicle from either team proceeds, more unscouted unregistered swimmers may move onto the list of scouted unregistered swimmers. An unregistered swimmer is considered a *scouted* when at least one vehicle has scouted the swimmer.

So in the rs2 mission the game is won at the first point in time when the majority of the currently known swimmers has been rescued by one team. Of course at that point there still may be undiscovered, unscouted unregistered swimmers, and the losing vehicle may in theory be able to still rescue more swimmers. For this reason, in the rs2 mission, the competion is finished when one team has won, even if the `finish_upon_win` parameter is set otherwise.

Tie-Breakers in the Rescue-Scout2 Mission

In the rs2 mission the number of presently known swimmers may be an even number. If the competition ends with each team having rescued the same amount of swimmers, the tie break comes down to the timestamp of the final swimmer rescue. The losing team would then be the one that was last to discover its final swimmer.

Win Flags and Finished Flags

The rescue manager can be configured to post one or more MOOS flags at the moment the competition has been won, or the has finished, set in the `winner_flag` or `finish_flag` configuration parameters. For example:

```
winner_flag = SAY_MOOS = $WINNER has won
```

With the above, if the `iSay` app is also running on the shoreside, the mission could be configured to make an audible declaration of the winner.

Another example:

```
finish_flag = RETURN_ALL = true
```

With the above, all vehicles will automatically transition to return home after the competition has finished. The leader, winner and finish flags support the two macros `$WINNER` and `$LEADER` expanded

to the name of the vehicle winning or leading at the time of the posting.

4.7 Dynamic New Swimmers and Rescues - For Testing

It is possible to inject *dynamic* swimmers during the course of a mission, after `uFldRescueMrg` has been started and loaded with a swim file. The reason for supporting this is to allow a user to simulate (a) the discovery of new swimmers (alerts), and (b) the message indicating that someone else has rescued a known swimmer. This is to simplify testing of these events during development.

A new swimmer is added when the rescue manager receives a message like:

```
XSWIMMER_ALERT = type=reg, x=31, y=193
```

Unregistered swimmers can also be added with:

```
XSWIMMER_ALERT = type=unreg, x=139, y=-73 (Note type=unreg)
```

In the rescue baseline mission, `pMarineViewer` is configured to allow mouse clicks adding registered and unregistered swimmers. Doing so may looks something like:

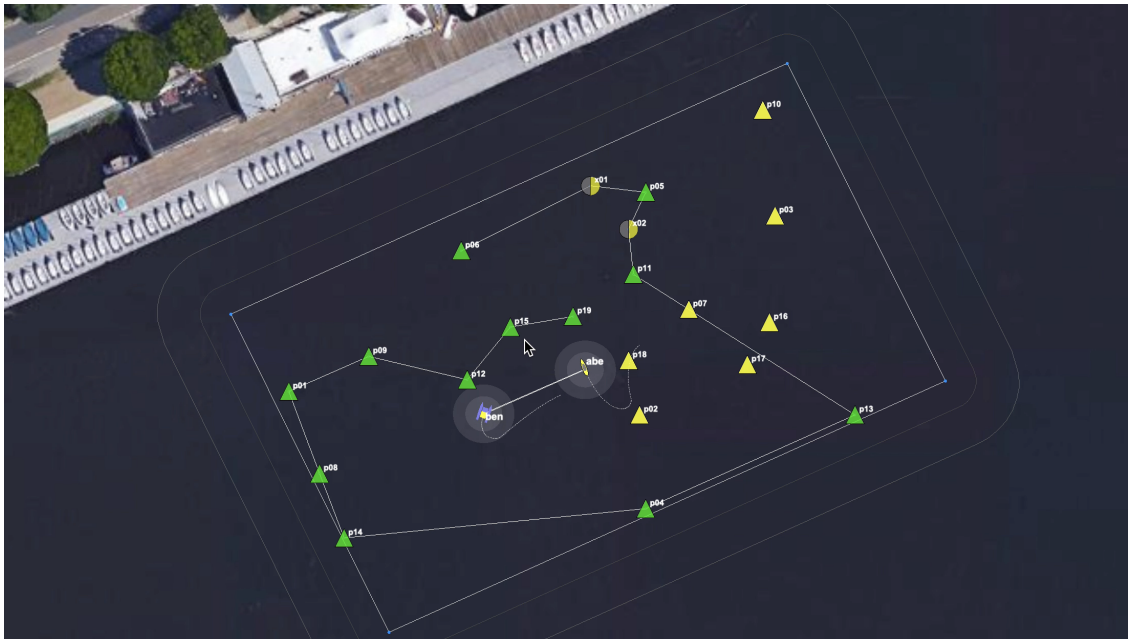

Figure 7: **Dynamic Swimmers**: In this simulation snippet, the viewer is configured to allow the user to add registered or unregistered swimmers at the location of the left or right mouse click. This allows the user to test either of these events in isolation.

Similar to adding new swimmers, the rescue manager allows for the simulation of a rescue made by "another" vehicle. This is done by posting a message such as:

```
XFOUND_SWIMMER = x=31, y=193
```

This will result in the rescue of the nearest currently-unrescued swimmer, as long as it is within 10

meters of the x-y point specified. The savior, or credit, rather than going to any named vehicle, is listed simply as "nature". This feature is a handy tool, when configured as in the rescue baseline mission, with the pMarineViewer configuration, to test how the path of the rescue vehicle is updated when a previously unrescued swimmer has become rescued. An example of this user intervention is shown in Figure 8.

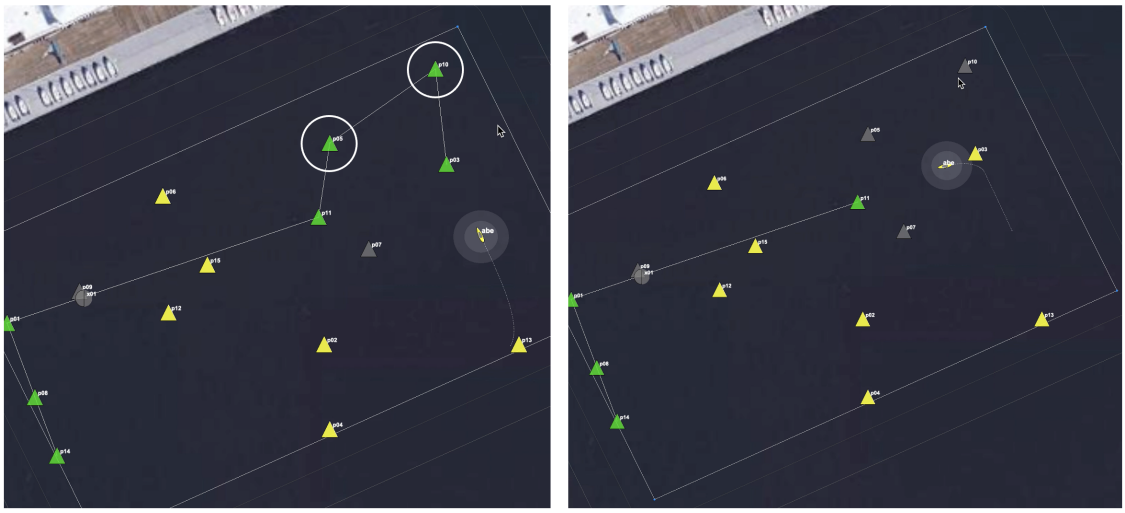

Figure 8: **Dynamic Rescue**: In this simulation snippet, the viewer is configured to add registered or unregistered swimmers at the location of the left or right mouse click. This allows the user to test either of these events in isolation.

5 Configuration Parameters of uFldRescueMgr

The following parameters are defined for uFldRescueMgr. A more detailed description is provided in other parts of this section. Parameters having default values are indicated so.

Listing 5.1: Configuration Parameters for uFldRescueMgr.

swim_file:	The file with ground truth information on swimmer location, ID and swimmer type. Section 3.1 and Section 3.3.
swimmer_color:	The default color of markers representing known swimmers, before they are rescued. The default is `dodger_blue`. In the rescue baseline mission it is set to `lime`. Section 4.3.
show_rescue_range:	If true, visual range circles are generated to be displayed around the moving vehicle to indicate the range within which a rescue will be made of a nearby swimmer. The default is true. Section 4.2.
rescue_range_transparency:	The transparency of the rescue range circles if they are rendered. The default is `0.1`, where `0` is completely transparent, and `1` is completely opaque. Section 4.2.

rescue_range_min:	The range between swimmer and vessel, within which a rescue request is guaranteed to be successful. The default is 25 meters. However, in the rescue baseline mission, it is set to 3 meters. Section 4.2.
rescue_range_max:	The range between swimmer and vessel, beyond which a rescue request is guaranteed to be unsuccessful. The default is 40 meters. However, in the rescue baseline mission, it is set to 5 meters. Section 4.2.
rescue_range_pd:	The probability success for a rescue request when the range between the simmer and the vessel is at or less than the rescue_range_min. The probability decrease to zero between rescue_range_min out to rescue_range_max, beyond which the probability is zero. Legal settings are in the range (0,1]. The default value is 1.0. Section 4.2.
leader_flag:	A flag posted when there is a change in the leader. Section 4.5.
finish_flag:	A flag posted when the competition has finished. Section 4.6.
winner_flag:	A flag posted when a winner has been cemented. Section 4.6.
finish_upon_win:	If true, the competition is declared finished when a winner has been declared (no change of leadership possible). The default is false. See Section4.6.

An Example MOOS Configuration Block

To see an example MOOS configuration block, enter the following from the command-line:

```
$ uFldRescueMgr --example or -e
```

This will show the output shown in Listing 2 below.

Listing 5.2: Example configuration of the uFldRescueMgr application.

```
1  ============================================================
2  uFldRescueMgr Example MOOS Configuration
3  ============================================================
4
5  ProcessConfig = uFldRescueMgr
6  {
7    AppTick   = 4
8    CommsTick = 4
9
10   // Common to all appcasting MOOSApps
11   term_report_interval = 0.4                  // default
12   max_appcast_events   = 8                    // default
13   max_appcast_run_warnings = 10               // default
14
15   swim_file     = file.txt
16   swimmer_color = dodger_blue
18
19   // Configuring visual preferences
20   show_rescue_rng = true                      // default
21   rescue_rng_transparency = 0.2               // default
```

```
22
23    // Sensor Config
24    rescue_rng_min   = 25                        // default
25    rescue_rng_max   = 40                        // default
26    rescue_rng_pd    = 0.5                       // default
27
28    // Event Flags
29    leader_flag = NEW_LEADER=$[LEADER]
30    winner_flag = NEW_WINNER=$[WINNER]
31    finish_flag = MISSION_COMPLETE=true
32    finish_flag = RETURN_ALL=true
33
34    finish_upon_win = false                      // default
35  }
```

6 Publications and Subscriptions for uFldRescueMgr

The interface for uFldRescueMgr, in terms of publications and subscriptions, is described below. This same information may also be obtained from the terminal with:

```
$ uFldRescueMgr --interface or -i
```

6.1 Variables Published by uFldRescueMgr

The primary output of uFldRescueMgr to the MOOSDB is the posting of requests for sensor information and the generation of results back to the vehicles.

- APPCAST: Contains an appcast report identical to the terminal output. Appcasts are posted only after an appcast request is received from an appcast viewing utility. Section 7
- NODE_COLOR_CHANGE_ABE: A message. Section
- PLOGGER_CMD: A message. Section 3.3.
- RESCUE_REGION: A convex polygon within which all swimmers are located. Section 3.3.
- RESCUED_SWIMMER: A message. Section 4.1.
- SCOUTED_SWIMMER_ABE: A message. Section 4.4.
- SWIMMER_ALERT_ABE: A message. Section 3.1.
- VIEW_MARKER: A marker representing the swimmer location. Section 4.3.
- VIEW_CIRCLE: A marker for conveying the sensor range around each vehicle. Section 4.3.
- VIEW_POLYGON: A polygon conveying the region of operation. Section 4.3.
- UFRM_WINNER: A message conveying the winner of the competition. Section 4.6.
- UFRM_LEADER: A message conveying the current leader of the competition. Section 4.5.
- UFRM_FINISHED: A message conveying whether or not the mission has been finished. Section 4.6.

6.2 Variables Subscribed for by uFldRescueMgr

The uFldRescueMgr application will subscribe for the following four MOOS variables:

- APPCAST_REQ: A request to generate and post a new apppcast report, with reporting criteria, and expiration.
- NODE_REPORT: Node reports are received for each vehicle in the mission, simulation or otherwise. They provide the name, type and location of the vehicle. They are published by pNodeReporter and are otherwise generally published by the vehicles and shared to the shoreside.
- RESCUE_REQUEST: Section 4.1.
- SCOUT_REQUEST: Section 4.4.
- XSWIMMER_ALERT: This message enables the dynamic addition of a new swimmer, in mid-mission. Section 4.7.
- XFOUND_SWIMMER: This message simulates the rescue of a swimmer, mid-mission. Section 4.7.

Command Line Usage of uFldRescueMgr

The uFldRescueMgr application is typically launched as a part of a batch of processes by pAntler, but may also be launched from the command line by the user. To see command-line options enter the following from the command-line:

```
$ uFldRescueMgr --help or -h
```

This will show the output shown in Listing 3 below.

Listing 6.3: Command line usage for uFldRescueMgr.

```
1  ============================================================
2  Usage: uFldRescueMgr file.moos [OPTIONS]
3  ============================================================
4
5  Options:
6    --alias=<ProcessName>
7        Launch uFldRescueMgr with the given process name.
8    --example, -e
9        Display example MOOS configuration block.
10   --help, -h
11       Display this help message.
12   --interface, -i
13       Display MOOS publications and subscriptions.
14   --version,-v
15       Display release version of uFldRescueMgr.
```

7 Terminal and AppCast Output

The uFldRescueMgr application produces some useful information to the terminal and identical content through appcasting. An example is shown in Listing 4 below. On line 2, the name of the app and the MOOS community is listed on the left. On the right, "0/0(1414) indicates there are no configuration or run warnings, and the current iteration of uFldRescueMgr is 783. Lines 4-12 convey the requested and prevailing rescue manager configuration settings.

In lines 13-19, the high level status of the mission is shown: the number of vehicles, leader (if any), and winner (if any). Lines 18 and 19 are often viewed as the defininitive indication of whether the

competition has been completed.

In lines 22-28, the stats *per vehicle* are shown. Since `abe` and `ben` are rescue vehicles, the may have a non-zero values in the `Swimmers Rescued` column. Likewise since `cal` and `deb` are the sole two scout vehicles, they will have non-zero totals for the `Scout Request` and `Scout Ties` columns and the `Swimmers Scouted` column. A scout request is the receipt of the `SCOUT_REQUEST` message. Some of these may be rejected by the rescue manager for arriving too soon after the previous request. Those that are not rejects are considered to be a "try".

Listing 7.4: Example `uFldRescueMgr` *console output.*

```
 1 ================================================================
 2 uFldRescueMgr shoreside                                  0/0(783)
 3 ================================================================
 4 ====================================
 5 RescueMgr Configuration
 6 ====================================
 7 rescue_rng_min: 3
 8 rescue_rng_max: 5
 9 rescue_rng_pd:  1
10 rescue_rng_show: true
11 transparency:   0.10
12 swim_file:      mit_rand.txt
13
14 ====================================
15 Vehicle Rescue Summary
16 ====================================
17 Total vehicles: 4
18 Leader vehicle: ben
19 Winner vehicle: ben
20 Mission Finished: true (0 remaining)
21
22 Vehi  Rescue  Rescue  Swimmers  Scout  Scout  Swimmers
23 Name  Reqs    Tries   Rescued   Reqs   Tries  Scouted
24 ----  ------  ------  --------  -----  -----  --------
25 abe   310     257     1         0      0      0
26 ben   309     257     8         0      0      0
27 cal   0       0       0         310    171    0
28 deb   0       0       0         310    171    3
29
30 ====================================
31 Swimmer Summary
32 ====================================
33 Name  ID    Type   Pos      State     Savior  Tries  Scouts  Time
34 ----  ----  -----  -------  --------  ------  -----  ------  -----
35 p01   id92  reg    48,-36   rescued   ben     1              0
36 p02   id45  reg    32,-24   rescued   abe     2              0
37 p03   id97  reg    34,-61   rescued   ben     2              0
38 p04   id08  reg    -14,-99  rescued   ben     1              0
39 p05   id62  reg    -1,-81   rescued   ben     2              0
40 x01   id48  unreg  -9,-23   swimming  -       0              441.7
41 x02   id43  unreg  43,-90   swimming  -       0              441.7
42 x03   id75  unreg  -12,-49  swimming  -       0              441.7
43 x04   id34  unreg  6,-20    rescued   ben     2      deb     0
44 x05   id25  unreg  53,-66   rescued   ben     1      deb     0
```

```
45 x06   id06  unreg  76,-39   rescued   ben     1      deb     0
46 x07   id28  unreg  -32,-78  swimming  -       0              441.7
47 x08   id63  unreg  87,-52   swimming  -       0              441.7
48 x09   id68  unreg  42,-77   rescued   ben     1              0
49
50 ================================================================
51 Most Recent Events (12):
52 ================================================================
53 [296.18]: Swimmer x04 has been rescued by ben!
54 [236.07]: Swimmer x06 has been rescued by ben!
55 [206.02]: Swimmer x05 has been rescued by ben!
56 [195.69]: Swimmer x09 has been rescued by ben!
57 [184.17]: Swimmer x06 has been scouted by deb!
58 [146.00]: Swimmer p04 has been rescued by ben!
59 [135.56]: Swimmer x05 has been scouted by deb!
60 [129.74]: Swimmer p05 has been rescued by ben!
61 [96.23]: Swimmer p03 has been rescued by ben!
62 [72.01]: Swimmer p01 has been rescued by ben!
63 [56.89]: Swimmer p02 has been rescued by abe!
64 [43.55]: Swimmer x04 has been scouted by deb!
```

In lines 30-48, the stats *per swimmer* are shown. Recall that registered swimmers have a name beginning with `'p'` and unregistered with `'x'`. the `ID` column shows the randomly generated ID alias presented to the vehicles, to hide any hints about the total number of unregistered swimmers. The `Type` column shows the type. The `Pos` column shows the x-y position. The `State` of a swimmer is either `swimming` or `rescued`. If rescued, the vehicle that made the rescue is shown in the `Savior` column. The `Tries` column shows how many swimmer `RESCUE_REQUEST` messages were needed before a rescue was granted. If an unregistered swimmer was scouted, it be shown in the `Scouts` column. In this example unregistered swimmer `x09` was rescued but not scouted. It can be inferred that the rescue vehicle `ben` came across this swimmer by coincident. The `Time` column indicates how long an unrescued swimmer has been unrescued.

Finally, in lines 50-64, recent events of the rescue manager are shown. These are typically successful rescues or scouts, showing the swimmer name and the vehicle responsible.