Gym-Aquaticus: An OpenAI Gym Reinforcement Learning Environment for MOOS-IvP-Aquaticus

Michael McCarrick
U.S. Naval Research Laboratory
Washington, DC

March 2021

Contents

1 Introduction **2**

2 Component Description **3**
2.1 gym-aquaticus 3
2.2 moos-ivp-rlagent 4

3 Installation **5**
3.1 Dependencies 5
3.2 Installation Steps 6
3.3 Docker 7

4 Configuration **8**

5 Caveats **9**

1 Introduction

Gym-Aquaticus is an OpenAI Gym compatible reinforcement learning (RL) environment built around the MOOS-IvP Aquaticus multi-player competition. Aquaticus is a unique game environment in that it can be played in both real-world competitions with human and robot (unmanned surface vehicles or USVs) participants and in a completely simulated environment on one or more computers. It is a capture-the-flag style competition played on the water between two opposing teams.

OpenAI Gym (https://gym.openai.com/) is a toolkit for developing and comparing reinforcement learning algorithms. In addition to providing a library of reinforcement learning tasks, Gym defines a standard interface to these environments that can be used to create compatible custom environments. By implementing the Gym interface, tasks in the Gym-Aquaticus environment can be trained using any compatible reinforcement learning algorithm, such as those in OpenAI Baselines (https://github.com/openai/baselines) or a more recent, updated fork, Stable Baselines (https://stable-baselines.readthedocs.io/). These high-quality RL implementations are well tested, well documented, and ready to apply.

Gym-Aquaticus is an outgrowth of an excellent first approach to adding reinforcement learning to Aquaticus, namely pLearn, by Arjun Gupta, now maintained by Michael Novitzky (https://github.com/mnovitzky/moos-ivp-pLearn). In pLearn, a python interpreter is embedded in the C++ IvP behavior. This allows the behavior to call python methods to control the helm using a neural network to predict the optimum course, given the current state. During training, it relies on parsing post-simulation log files after each episode to evaluate and improve the performance of the neural network. The learning algorithm is a custom-built DeepQ Neural Network.

Gym-Aquaticus takes a complementary approach. In our case the C++ behavior is a very simple function that accepts helm commands (speed, heading) from the controlling python routine via MOOS mail messages. The python routines implement the Gym interface to allow any given RL algorithm to sample the game's state space, determine the optimum action, and send the associated helm command using pymoos (https://github.com/russkel/python-moos), a python implementation of a MOOSAsyncCommClient. Since the state and associated reward function are sampled at every step (e.g. every 400 msec in simulated time or typically >100 times per episode), the learning algorithm gets much finer-grained information on the result of its action and converges quickly to optimum behavior.

It is important to note that this is a very early and incomplete starting point for RL research in the Aquaticus environment using OpenAI Gym and Stable Baselines algorithms. The current release does successfully learn a single, simple task (capture the flag with a naïve defender), but a great deal of work remains to be done.

2 Component Description

Two major components comprise Gym-Aquaticus, `gym-aquaticus` and `moos-ivp-rlagent`, corresponding to the python and C++ code elements. The two components are maintained in separate git repositories because they each follow the directory layout and content specified for their respective function.

2.1 gym-aquaticus

The `gym-aquaticus` source code tree is an implementation of an OpenAI Gym environment customized for the Aquaticus game. It follows the layout described in https://github.com/openai/gym/blob/master/docs/creating-environments.md. There are two primary classes defined in this package:

`AquaticusEnv` is derived from `gym.Env`. `AquaticusEnv` follows the API required for an OpenAI Gym environment, exposing several critical methods, `reset()`, `step()`, and `close()`, that will be called by any RL algorithm. `AquaticusEnv` contains an instance of a particular actor, `AquaticusAttacker`, to implement the RL task defined there. It is anticipated that other RL tasks (e.g. return the flag safely to score points) would be implemented in subsequent actor classes with different observations spaces and reward functions, but all could use the same interface to the RL training algorithm, `AquaticusEnv`. The API methods are:

- `reset()`
  - start an Aquaticus episode via shell script
  - return the initial state observation
- `step(action)`
  - take the desired action via helm command
  - `sleep(step_time)` to allow the game to advance
  - return the new observation and reward
- `close()`
  - terminate all MOOS processes (end the episode) via shell script

`AquaticusAttacker` is derived from `pymoos.comms`, which is a python implementation of the `MOOSAsyncCommClient` class from `libMOOS`. `AquaticusAttacker` defines the action space, observation state space, and reward function necessary to implement a single RL task, namely, to capture the enemy flag while avoiding defenders. All of these attributes and methods are clearly defined in the python source module and can easily be modified to experiment with alternate definitions. The important methods are:

- `_on_connect()`

- register for MOOS variables (ownship, x, y, enemy x, y, tagged state, etc.)

- `_on_mail()`
  - update internal state variables with new values from the MOOSDB
- `get_observation()`
  - return the current observation state, an instance of `gym.spaces.Box()`
- `get_reward()`
  - return the calculated reward function at the current state
- `take_action()`
  - use `MOOSCommClient.Notify()` to post the desired action (speed, heading) to the MOOSDB

2.2 moos-ivp-rlagent

The `moos-ivp-rlagent` source code tree is an implementation of a MOOS-IvP extension as described in https://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=Helm.MOOSIvPExtend. It defines a standard IvP-Helm behavior, accepting speed and heading updates from the `AquaticusAttacker` RL agent via the MOOSDB and producing a valid IvP objective function. The important components are described below.

`BHV_RLAgent` is derived from `IVPBehavior`. It has the following features:

- Registers for helm command variables from AquaticusAttacker, `RLA_SPEED` and `RLA_HEADING`
- Produces a valid IvP objective function when active
- Completes (goes idle) when the goal is reached, a tag is applied, or ownship goes out-of-bounds
- Restarts (goes active) when a tag is cleared or the enemy flag is returned

`RLMonitor` is derived from `AppCastingMOOSApp`. This simple AppCast application can be used to monitor the reinforcement learning internal variables during an Aquaticus game in uMACView.

`missions/red_vs_blue` is a folder containing all of the various shell scripts, `.moos` configuration files, and `.bhv` behavior files necessary to run the Aquaticus game. These are mostly copied from the equivalent missions directory in pLearn, but modified slightly for the Gym-Aquaticus environment. Also contained in this directory are the python scripts that do the actual RL training and testing:

- `dqn_train.py` is a simple example script that creates an OpenAI Gym environment from Gym-Aquaticus, creates a training model using the Stable Baselines DQN (Deep-Q neural network) RL algorithm with default hyperparameters, and trains the model over 100,000 steps, which takes about 4.2 hours. By default, the only output during training is a summary of performance after each episode. If the user creates an empty file called `show` in this directory, then each episode will be rendered using pMarineViewer during training. The `show` file can be created or deleted any time during training to enable or disable viewing. It doesn't affect the speed of training. At completion, the trained model is saved as `dqn_latest.zip`.
- `dqn_test.py` runs a complete Aquaticus game using the previously-saved model. Whereas the episodes (games) started in `dqn_train.py` are immediately terminated once the attacking ship captures the flag, gets tagged, or goes out of bounds, the game started in `dqn_test.py` will continue playing out with the attacking ship cycling through different behaviors depending on the states defined in its `.bhv` file.

3 Installation

This section is incomplete at this time but provides some hints and tips for installation. Gym-Aquaticus is currently distributed as two separate git repositories or equivalent zip files. These are not yet publicly available, but may potentially be obtained from an NRL contact.

Gym-Aquaticus requires a working installation of MOOS-IvP and MOOS-IvP-Aquaticus as well as the specific dependencies described later. Please see the excellent documentation at https://oceanai.mit.edu/moos-ivp/ and https://oceanai.mit.edu/aquaticus/ for help installing those components.

Assuming you have obtained `gym-aquaticus-master.zip` and `moos-ivp-rlagent-master.zip`, unzip these file into a directory of your choice. This should be the same directory that holds `moos-ivp` and `moos-ivp-aquaticus`. After unzipping, please rename the folders to `gym-aquaticus` and `moos-ivp-rlagent`, i.e., remove the `-master` artifact from the git repository zip file.

3.1 Dependencies

Gym-Aquaticus currently requires both OpenAI Gym and Stable Baselines for training. Stable Baselines requires TensorFlow 1.15 (or at least >1.8.0 and <2.0.0, but 1.15 is the last 1.xx version). TensorFlow 1.15 requires python 3.5-3.7. The standard python version installed in Ubuntu 18.04 LTS is 3.6.9, which works fine, but Ubuntu 20.04 LTS includes python 3.8.5 which unfortunately won't work with TensorFlow 1.15. For Ubuntu 20.04, python 3.7.10 can be installed by adding the “deadsnakes” PPA (personal package archive) to the system:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt install python3.7 python3.7-dev
$ python3.7 -V
Python 3.7.10
```

The python package installer, `pip`, is also needed. The system version can be installed using apt.

```
$ sudo apt install python3-pip
$ python3 -m pip -V
pip 20.0.2 from /usr/lib/python3/dist-packages/pip
```

Some version of macOS include an appropriate release of python3. Alternatively, the most recent compatible version, python 3.7.10, can be installed with homebrew or otherwise.

```
$ brew install python@3.7
$ /usr/local/opt/python@3.7/bin/python3 -V
Python 3.7.10
```

For minimum confusion, it is useful to employ python’s virtual environment feature to create a separate python installation just for Gym-Aquaticus. This separates user-installed versions of the various python libraries from the system versions. Also, the specific python version which is used to create the virtual environment becomes the default in that environment, so there is no need to adjust your `PATH` to pick up the right version.

3.2 Installation Steps

Create and activate a virtualenv using python 3.7:

```
$ python3.7 -m virtualenv venv # you can name the environment anything

  -OR- for macOS with a non-system brew installation:

$ /usr/local/opt/python@3.7/bin/python3 -m virtualenv venv

$ source venv/bin/activate # make sure to activate the virtual
                           # environment before continuing
$ (venv) python -V
Python 3.7.10
```

Install prerequisites for Gym and Stable Baselines:

```
$ (venv) pip install -U pip # the newest pip is required for later steps
$ (venv) pip install numpy matplotlib scipy colorama
$ (venv) pip install tensorflow==1.15 keras==2.0.8
```

Install OpenAI-Gym:

```
$ (venv) pip install gym[all] # ignore mujoco-py errors, MuJoCo is not installed
```

Install Stable Baselines:

```
$ sudo apt install libopenmpi-dev
$ (venv) pip install stable-baselines[mpi]
```

Install and build `moos-ivp-rlagent`:

```
$ (venv) unzip moos-ivp-rlagent-master.zip
$ (venv) mv moos-ivp-rlagent-master moos-ivp-rlagent
$ (venv) cd moos-ivp-rlagent
$ (venv) ./build.sh
```

Install `gym-aquaticus`:

```
$ (venv) unzip gym-aquaticus-master.zip
$ (venv) mv gym-aquaticus-master gym-aquaticus
$ (venv) pip install -e gym-aquaticus
```

Add `moos-ivp-rlagent` environment variables to `.bashrc`, `.bash_profile`, or equivalent (assuming installation is in your `$HOME` directory):

```
...
export PATH=$PATH:$HOME/moos-ivp/bin:$HOME/moos-ivp-aquaticus/bin
export PATH=$PATH:$HOME/moos-ivp-rlagent/bin
export IVP_BEHAVIOR_DIRS=$HOME/moos-ivp/lib:$HOME/moos-ivp-aquaticus/lib
export IVP_BEHAVIOR_DIRS=$IVP_BEHAVIOR_DIRS:$HOME/moos-ivp-rlagent/lib
...
```

3.3 Docker

The Aquaticus-Gym environment can also be run as a Docker image. A suitable Docker file can be found in `moos-ivp-rlagent/Docker`. However, because we don't have a publicly accessible repository, the build process is a little different.

First, install the `moos-ivp-rlagent` and `gym-aquaticus` trees as above by unziping and renaming the appropriate zip files. In this case we don't need any of the dependencies (python 3.7, moos-ivp, etc.) since the Docker image will contain all of those. Next, copy the

Docker file up one level before building. This is required because Docker build commands can't access "side directories" (in this cases we would need `../gym-aquaticus`). Finally, build the image and run a container.

```
$ unzip moos-ivp-rlagent-master.zip
$ mv moos-ivp-rlagent-master moos-ivp-rlagent
$ unzip gym-aquaticus-master.zip
$ mv gym-aquaticus-master gym-aquaticus
$ cp moos-ivp-rlagent/Docker .
$ docker build -t rlagent:1.0 .
$ docker run --name rlagent -it rlagent:1.0 /bin/bash
$ docker exec -it rlagent /bin/bash
```

4 Configuration

Important configuration files and shell scripts include the following:

`gym-aquaticus/gym_aquaticus/envs/config.py` This file contains the important Aquaticus parameters that are needed by the reinforcement learning agent. The `moos_server`, `moos_port`, boundaries, and flag locations must all match those defined in the Aquaticus configuration files. This file also contains constants used in defining the action space, the reward function, and the simulation time step.

`moos-ivp-rlagent/missions/train.sh` This script is called by the RL environment to start a new episode every time `reset()` is invoked. It runs `launch_train.sh` with a timewarp of 4 and then uses uPokeDB to deploy the USVs and start the simulation.

`moos-ivp-rlagent/missions/test.sh` This script is used in the testing environment to run a previously-trained model. It runs `lauch_test.sh` which is nearly identical to `launch_train.sh` except that it starts a different script to launch the USVs and starts pMarineViewer by default.

`moos-ivp-rlagent/bin/killAllMOOS.sh` This script is called from the `close()` method in the AquaticusEnv class to terminate an Aquaticus episode. The script provided simply issues a `kill -9` (non-ignorable kill) to all processes it finds that look like MOOS processes. We found that `ktm`, the more proper way to terminate MOOS processes, is considerably slower and didn't always terminate everything, leaving a few `pAntler` or other processes hanging. After many iterations, this causes the system to become unstable or use excessive CPU time.

`moos-ivp-rlagent/missions/red_vs_blue/meta_m200.bhv` and `moos-ivp-rlagent/missions/red_vs_blue/meta_m200_train.bhv` These MOOS behavior files include a section for the `BHV_RLAgent` behavior. The required parameters include the enemy flag location (`flag_x`, `flag_y`) and the capture radius. The `endflag` in this section causes a `FLAG_GRAB_REQUEST` to be issued when the behavior completes, that is when the USV is within `capture_radius` of the enemy flag position.

The `moos-ivp-rlagent/missions` directory was largely copied from the corresponding pLearn `simulation_engine` directory. It includes the README file from that distribution, which describes a few more details about the included files. There are a variety of options in the scripts that are not used in the current RL environment, but are left in place for future use.

5 Caveats

This is **alpha** code, not ready for wide release. This very early release is intended to be used by people familiar with the MOOS-IvP environment who want to experiment with the OpenAI Gym interface and Stable Baselines RL algorithms.

When the RL training sequence runs, hundreds or even thousands of Aquaticus simulations are started and terminated. This leads to occasional problems in process startup/termination or networking. The AquaticusEnv class has some methods that check for and try to recover from the most common problems. This mostly works fine, but occasionally an exception is thrown that causes the python interpreter to abort. Typically those exceptions are related to low-level MOOS network activity (`XPCException`). With the timing, sleeps, and moos_timewarp in the current distribution, we have successfully run 200,000 iterations of training in a number of different environments (macOS, Ubuntu, Docker, etc.) without any failures. However, if the timing is adjusted (shorter sleeps, higher timewarp) that may change. Feel free to experiment!