

Contents

1	/	3
1.1	README.md	3
1.2	agents.py	5
1.3	coach.py	10
1.4	directors.py	18
1.5	env.py	19
1.6	params.py	24
1.7	requirements.txt	25
1.8	texify.py	26
2	DASH	28
2.1	app.py	28
2.2	coach_integration.py	34
2.3	dash_utilities.py	41
2.4	html_objects.py	42
3	DASH/Documentation	46
3.1	interface.txt	46
3.2	network_view.py	48
4	DASH/assets	50
4.1	typography.css	50
5	examples/MAInspection	53
5.1	Readme.md	53
5.2	app.py	54
5.3	train_agents.py	56
5.4	train_director.py	59
6	examples/MAInspection/Environments	62
6.1	MAIagents.py	62
6.2	MAIcoach.py	64
6.3	env.py	69
7	examples/MAInspection/Experiment	85
7.1	dash_app.yaml	85
7.2	dash_app_template.yaml	86
7.3	train_agents.yaml	87
7.4	train_director.yaml	88
8	examples/MAInspection/assets	89
8.1	typography.css	89
9	examples/Multipolicy	92
9.1	Readme.md	92
9.2	app.py	93
9.3	train_agents.py	95
9.4	train_director.py	98
10	examples/Multipolicy/Experiment	101
10.1	dash_app.yaml	101
10.2	dash_app_template.yaml	102
10.3	train_agents.yaml	103
10.4	train_director.yaml	104

11	examples/Multipolicy/assets	105
11.1	typography.css	105
12	utilities	108
12.1	PZParams.py	108
12.2	PZWrapper.py	110
12.3	iotools.py	113
12.4	planning.py	117
12.5	timelines.py	121
12.6	tools.py	126

1 /

1.1 README.md

```
1 # Controller-Agnostic Orchestration Architecture (COACH)
2 ## A Framework for Training Semi-Autonomous Agents and Directors Leveraging Existing Simulations
3
4 COACH is a system to turn (PettingZoo)[https://pettingzoo.farama.org/index.html] (PZ) compatible
  multiagent Gymnasium environments into agent orchestration and planning problems, using a
  director/actor hierarchical framework. Given any PZ environment and a set of agents trained to
  execute actions on that environment given asynchronous direction in terms of a __plan__, the COACH
  repo provides a wrapper to turn the problem into a planning simulation and a wrapper to turn such
  planning simulations into PZ compatible orchestration environments for training director agents. In
  addition, it provides a prefab agents and an interactive plan visualization system using DASH.
5
6 This library can be used as is, but has been designed as a starting point for interactive planning
  problems.
7
8 
9
10 ## Terminology:
11 A lot of the terminology around RL becomes contested when you move into the multiagent setting. For this
  library we will nail down the following conventions, with the explicit understanding that they are not
  universal. We have tried, as much as possible, to keep the language as close to any standard we
  could find as possible.
12
13 * Simulation - A (multiagent) simulation environment conforming to the (PettingZoo
  interface)[https://pettingzoo.farama.org/index.html]
14 * Agent - A piece of code that takes observations and returns actions.
15 * Actor - An agent assigned to a specific role in multiagent simulation, and equipped with an instruction
  set (interface) for directing the actor. Actors are "interactive" or "directable" agents.
  Importantly, actors may be RL policy based, have multiple policies, use good-old-fashioned-ai, or
  be generated using any other method.
16 * Course Of Action (COA) - A set of directions for a particular actor, tagged to specific times in the
  language of the agents interface.
17 * Plan - A set of courses of action, one for each role in the simulation.
18 * COACH Environment - An environment for planning problems: Given a simulation and a set of actors
  filling the required roles, a directing agent generates a course of action for each agent to
  follow. The simulation is then run forward a number of steps (fixed, agent dependent, or agent
  dependent with blackout conditions), and the courses of action may be updated.
19 * Director - An agent that takes observations recorded by actors, and generates a courses of action.
20 * Trajectory - Action/observation/rewards for all agents over a period of time.
21
22 # Getting started
23
24 For a good introduction, take a look at the examples provided in the examples folder. They give a
  demonstration of how to train and deploy a multi-policy actor with a director choosing which policy
  to use at a given time.
25
26 ## Examples:
27
28 ### Multipolicy Director
29
30 In this example, we use StableBaselines3 and Supersuit to train two policies on the Waterworld
  environment from PettingZoo. One we train in a sparse food environment as an "explorer" agent, the
  other we train in a dense food and dense poison environment as a "dense" avoider agent. Finally, we
  train a director to select the policies each agent should use for the next 10 turns given their
  current observations.
31
32 In this example we see how to train policies on a PettingZoo environment, use those policies and that
  PettingZoo environment to establish a planning problem using the coach environment. We then set up
  a communications schedule to allow communication every 10 turns and train the director agent to
  select between the policies.
33
34 Finally, we construct a Dash app to display our solution.
35
36 This example uses all most entirely default functionality. For more advanced functionality see the
  MAInspection example.
37
38 ### Multiagent Inspection
39
40 This example shows how to solve a simple satellite inspection problem in two steps:
41 * First we train an agent not to solve the problem, but to use the environment to train an agent to go
  from one waypoint to another.
42 * After this "waypointer"
  agent is trained the director is then trained to select waypoints that easily solve the problem.
43
44 This example also demonstrates how to create custom agents, a custom coach environment, custom
```

```

trajectory classes and custom trajectory visualizations.
47
48 ## Overview of Environment Files:
49
50 * `coach.py` - Contains the class that wraps a PZ env, turning it into a planning environment.
51 ...
52 COACHEnvironemnt(
53     env_creator: callable,
54     parameter_generator:BaseParameterGenerator,
55     agents=dict(),
56     fill_random=True,
57     comm_schedule:Timeline=None,
58     seed=6234235,
59     TrajectoryClass = Trajectory,
60 )
61 ...
62 * * `env_creator` - A function that creates a parallel PettingZoo env.
63 * * `parameter_generator` - A parameter generator object. Simulations call the parameter generator for
    samples. The BaseParameterGenerator simply returns the same parameters each time, the
    RandomParameterGenerator returns the same parameters but different seeds.
64 * * `agents` - A dictionary of `role:agent` pairs. If not specified the DefaultActor agent will be used.
65 * * `fill_random` - If `True` any agent roles not specified above will be set to the RandomActor agent.
66 * * `comm_schedule` - A `utilities.planning.CommunicationsSchedule` object that determines contact times
    and blackout times for director/agent communication.
67 * * `seed` - Random seed.
68 * * `TrajectoryClass` - Trajectories save agent history. By default all actions and step returns are
    saved, but if other information is required a different class can be passed.
69
70 * `env.py` - PettingZoo env wrapping COACHEnvironemnt for training director agents.
71 ...
72 COACH_PettingZoo(env_creator: callable, COACHEnvClass: COACHEnvironment)
73 ...
74 * * `env_creator` - A function that creates a parallel PettingZoo env.
75 * * `COACHEnvClass` - COACHEnvironemnt class that will be wrapped.
76
77 ## Overview of Auxiliary Files
78 * `agents.py` - Library of agents, sample agent classes include RandomActor, DefaultActor and
    SB_PPPOPoliciesActor
79 * `params.py` - Standard MarcoPolo params functionality for evolving parameters
80 * `utilities.iotools.py` - Tools for saving and reloading
81 * `utilities.planning.py` - Holds timeline classes specifically for COACH, including
    `CommunicationsSchedule` and `COA`
82 * `utilities.PZParams.py` - Parameter file for environments from the PettingZoo library
83 * `utilities.PZWrapper.py` - Wrapper for environments from the PettingZoo library
84 * `utilities.timelines.py` - Timeline classes
85 * `utilities.tools.py` - Miscellaneous tools used by parts of the coach env.
86
87 * `DASH` - Interactive Course Of Action viewer. See Multipolicy Example for more.
88 * `examples` - Examples of how to use the coach environment.

```

1.2 agents.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import sys
16
17 from utilities.timelines import TimelineEvent
18 from utilities.planning import COA
19 from numpy.random import PCG64DXSM, Generator
20 import numpy as np
21 import copy
22 import logging
23 from gymnasium.spaces import Box
24
25 logger = logging.getLogger(__name__)
26
27 from collections import OrderedDict
28 from stable_baselines3 import PPO
29 import sys, inspect
30
31 #####
32 ##
33 ## Interfaces
34 ##
35 #####
36
37 class ActionBox(Box):
38     def __init__(self, low=[], high=[], shape=[], default=[], description=None):
39         if len(low)>0:
40             super().__init__(low=low, high=high, shape=shape)
41         else:
42             self.low = []
43             self.high = []
44
45             self.description = description
46
47             if len(default) == len(low):
48                 self.default = default
49             elif len(default) == 0:
50                 default = []
51                 for l, h in zip(low, high):
52                     # If the bounds are finite, take the average.
53                     if np.isfinite(l) and np.isfinite(h):
54                         default.append((l+h)/2)
55                     # If one is infinite, we're going to assume the other is the default value
56                     elif np.isinf(l) and np.isfinite(h):
57                         default.append(h)
58                     elif np.isfinite(l) and np.isinf(h):
59                         default.append(l)
60                     ## Otherwise, assume 0
61                     else:
62                         default.append(0)
63
64             self.default = default
65         else:
66             raise Exception("Action Box has default values of length different than box.")
67
68     def __repr__(self) -> str:
69         return f"{self.description}: low {self.low}, high {self.high}"
70
71     def items(self):
72         actions = []
73         for i in range(len(self.low)):
74             actions.append(
75                 ActionBox(
76                     low = np.array([self.low[i]]),
77                     high = np.array([self.high[i]]),
78                     shape = (1,),
```

```

79         description = self.description[i]
80     )
81 )
82
83     return actions
84
85     def equals(self, other):
86         # All default interfaces are the same
87         if not (type(other) is type(self)):
88             return False
89         if not all([a==b for a,b in zip(self.low, other.low)]):
90             return False
91         if not all([a==b for a,b in zip(self.high, other.high)]):
92             return False
93         if not all([a==b for a,b in zip(self.shape, other.shape)]):
94             return False
95         if not all([a==b for a,b in zip(self.default, other.default)]):
96             return False
97
98     return True
99
100
101 class TrivialInterface:
102     def __init__(self, role, env):
103         self.env_observation_space = copy.deepcopy(env.observation_space(role))
104         self.env_action_space = copy.deepcopy(env.action_space(role))
105
106         self.action_space = self.env_action_space
107         self.observation_space = self.env_observation_space
108
109         self.action_dictionary = {"default": ActionBox()}
110
111         self.name = self.__class__.__name__
112
113     def action_names(self):
114         return list(self.action_dictionary.keys())
115
116     def get_action_descriptions(self):
117         # Return human readable description of the sceion parameters.
118         return self.action_description
119
120     def get_action_dictionary(self):
121         # Return the dctionary of possible actions with their parameter
122         # spaces
123         return self.action_dictionary
124
125     # We may want to verify and error check that an interface matches an expected
126     # interface, espeacilly if the interface takes parameters.
127     def equals(self, other):
128         # All default interfaces are the same
129         if type(other) is type(self):
130             return True
131         return False
132
133
134 class DefaultInterface(TrivialInterface):
135     def __init__(self, role, env, max_action_len=None):
136         super().__init__(role, env)
137
138         if not max_action_len:
139             max_action_len = np.inf
140
141         # The burn is "legnth" + "action space"
142         burn = ActionBox(
143             low=np.concatenate([(0,), self.env_action_space.low]),
144             high=np.concatenate([(max_action_len,), self.env_action_space.high]),
145             shape=(1 + np.prod(self.env_action_space.shape),),
146             description = ["Length"] + ["Unknown"] * len(self.env_action_space.low)
147         )
148
149         self.action_dictionary = {"burn": burn}
150
151     def equals(self, other):
152         # All default interfaces are the same
153         if type(other) is type(self):
154             return self.action_dictionary["burn"].equals(self.action_dictionary["burn"])
155
156         return False
157
158
159 class RandomActionInterface(TrivialInterface):

```

```

160     def __init__(self, role, env):
161         super().__init__(role, env)
162
163         self.action_dictionary = {"random": ActionBox()}
164
165
166 class SBPolicyInterface(TrivialInterface):
167     def __init__(self, role, env, n_policies, max_action_len):
168         super().__init__(role, env)
169
170         one_hot_low = [0]*n_policies
171         one_hot_high = [1]*n_policies
172         desc = ["Action Length"] + [f"policy_{i}" for i in range(n_policies)]
173
174         policies = ActionBox(
175             low=np.array([0] + one_hot_low),
176             high=np.array([max_action_len] + one_hot_high),
177             shape=(len(one_hot_low) + 1,),
178             description=desc
179         )
180
181         self.action_dictionary = {"Policies": policies}
182
183     def equals(self, other):
184         if type(other) is type(self):
185             return self.action_dictionary["Policies"].equals(other.action_dictionary["Policies"])
186         return False
187
188
189 #####
190 ##
191 ## Actors
192 ##
193 #####
194
195
196 class BasicActor:
197     InterfaceType = TrivialInterface
198
199     def __init__(self, role, reference_interface:TrivialInterface = None):
200         self.role = role
201         self.reference_interface = reference_interface
202
203     def action_names(self):
204         return list(self.interface.action_dictionary.keys())
205
206     def get_action_dictionary(self):
207         # Return the dctionary of possible actions with their parameter
208         # spaces
209         return self.interface.action_dictionary
210
211     def get_action_descriptions(self):
212         # Return human readable description of the sceion parameters.
213         return self.interface.action_description
214
215     def get_action(self, obs, t, mean_mode=False):
216         return self.none_action, {"acting": False, "coa_done": True}
217
218     def process_observations(self, obs, t):
219         # Agents may communicate different information than what they observe
220         return obs
221
222     def update_coa(self, coa):
223         self.coa.add_timeline_to(coa, allow_dup_labels=False)
224
225     def reset(self, env):
226         self.env = env
227         self.interface = BasicActor.InterfaceType(self.role, env)
228         if self.reference_interface is not None:
229             if not self.interface.equals(self.reference_interface):
230                 raise Exception("Actor interface does not match reference interface.")
231
232         self.coa = COA()
233
234         self.none_action = np.zeros(self.interface.action_space.shape)
235
236
237 class DefaultActor(BasicActor):
238     InterfaceType = DefaultInterface
239
240     def __init__(self, role, max_action_len=None):

```

```

241         super().__init__(role)
242         self.max_action_len = max_action_len
243
244     def __str__(self):
245         return f"DefaultActor: {self.role}"
246
247     def __repr__(self):
248         return self.__str__()
249
250     def get_action(self, obs, t, mean_mode=False):
251         if t in self.coa.timeline.keys():
252             e = self.coa.get(time = t, label = "burn")
253             self.burn_time = np.floor(e.parameters[0])
254             self.current_burn = np.array(e.parameters[1:]).reshape(
255                 self.interface.action_space.shape
256             )
257
258             time_stamps = list(self.coa.timeline.keys())
259             if len(time_stamps) == 0:
260                 coa_done = True
261             else:
262                 last_t = max(time_stamps)
263                 last_duration = self.coa.get(time = last_t, label= "burn").parameters[0]
264
265                 if t >= last_t + last_duration:
266                     coa_done = True
267                 else:
268                     coa_done = False
269
270             if self.burn_time > 0:
271                 self.burn_time -= 1
272                 return self.current_burn, {"acting": True, "coa_done": coa_done}
273             else:
274                 return self.none_action, {"acting": False, "coa_done": coa_done}
275
276     def reset(self, env=None):
277         super().reset(env)
278
279         self.interface = DefaultActor.InterfaceType(self.role, env, max_action_len=self.max_action_len)
280
281         self.current_burn = None
282         self.burn_time = 0
283
284
285 class RandomActor(BasicActor):
286     InterfaceType = RandomActionInterface
287
288     def __init__(self, role, seed=23143):
289         super().__init__(role)
290         self.seed = seed
291
292     def __str__(self):
293         return f"RandomActor: {self.role}"
294
295     def __repr__(self):
296         return self.__str__()
297
298     def get_action(self, action, parameters):
299         r = self.np_random.uniform(
300             low=self.interface.action_space.low, high=self.interface.action_space.high
301         )
302
303         return r, {"acting": True, "coa_done": False}
304
305     def reset(self, env):
306         super().reset(env)
307
308         self.interface = RandomActor.InterfaceType(self.role, env)
309         self.np_random = Generator(PCG64DXSM(seed=self.seed))
310
311
312 class SB_PP0PoliciesActor(BasicActor):
313     InterfaceType = SBPolicyInterface
314
315     def __init__(self, role, policy_paths: dict, max_action_len: int, interface:SBPolicyInterface=None):
316         super().__init__(role)
317
318         self.max_action_len = max_action_len
319
320         self.policies = dict()
321         for policy, model_path in policy_paths.items():

```



```

322         # For some reason SB doesn't want to zip on the end
323         if model_path[-3:] == "zip":
324             model_path = model_path.split(".")[0]
325             self.policies[policy] = PPO.load(model_path)
326
327         self.policy_names = list(self.policies.keys())
328
329     def __str__(self):
330         return f"PPOPoliciesActor: {self.role}, Policies: {self.policies.keys()}"
331
332     def get_action(self, obs, t, mean_mode=False):
333         bad_command = False
334         acting = False
335         coa_done = False
336
337         if t in self.coa.timeline.keys():
338             ## Start new policy
339             e = self.coa.get(time = t, label = "Policies")
340             self.time_remaining = e.parameters[0]*self.max_action_len
341             logger.debug("Agent: start new policy %s", e) # DEBUG
342             logger.debug("policy: %s, timeleft: %s",
343                         self.policy_names[int(np.argmax(e.parameters[1:]))], self.time_remaining) # DEBUG
344
345             self.current_policy = self.policy_names[int(np.argmax(e.parameters[1:]))]
346
347             # self.next_waypoint_abs = np.array([7.08763559, -25.61246231, -167.51736098])
348             if (self.current_policy is not None) and (self.time_remaining > 0):
349                 acting = True
350                 action, _ = self.policies[self.current_policy].predict(obs, deterministic=True)
351                 self.time_remaining += -1
352             else:
353                 action = copy.copy(self.none_action)
354
355             if self.time_remaining == 0:
356                 coa_done = True
357
358         return action, {
359             "acting": acting,
360             "coa_done": coa_done,
361             "bad_command": bad_command,
362         }
363
364     def reset(self, env):
365         super().reset(env)
366         self.time_remaining = -1
367         self.current_policy = None
368
369         self.interface = SB_PP0PoliciesActor.InterfaceType(
370             self.role,
371             env,
372             n_policies=len(self.policy_names),
373             max_action_len=self.max_action_len
374         )
375
376         if self.reference_interface is not None:
377             if not self.interface.equals(self.reference_interface):
378                 raise Exception("Actor interface does not match reference interface.")
379
380 classes = [
381     cls_obj
382     for cls_name, cls_obj in inspect.getmembers(sys.modules[__name__])
383     if inspect.isclass(cls_obj) and cls_obj.__module__ == __name__
384 ]
385
386 Interfaces = {}
387 for c in classes:
388     # BasicActor
389     if issubclass(c, TrivialInterface):
390         Interfaces[c] = []
391
392 Agents = {}
393 for c in classes:
394     # BasicActor
395     if issubclass(c, BasicActor):
396         Interfaces[c.InterfaceType].append(c)
397         Agents[c.__name__] = c

```

1.3 coach.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from typing import Any
16 import numpy as np
17 import copy
18 import logging
19 logger = logging.getLogger(__name__)
20 import argparse
21
22 import matplotlib.pyplot as plt
23 from numpy.random import PCG64DXSM, Generator
24
25 import agents as agents
26 from agents import (
27     BasicActor,
28     DefaultActor,
29     RandomActor
30 )
31
32 from utilities.timelines import Timeline, Timelines
33 from utilities.planning import (
34     COA,
35     CommunicationSchedule,
36     BaseParameterGenerator,
37     SeededParameterGenerator,
38     State,
39     Trajectory,
40     Telemetry,
41 )
42
43 from matplotlib.backends.backend_agg import (
44     FigureCanvasAgg as FigureCanvas, # type: ignore[import]
45 )
46
47 # %%
48
49 def get_env_class(args: argparse.Namespace):
50     """Returns the class to use, based on input arguments
51
52     Parameters
53     -----
54     args: argparse.Namespace
55         arguments that were passed to the `main()` function
56
57     Returns
58     -----
59     class
60         the class to use in creating env objects
61     """
62
63     return COACHEnvironment
64
65 class COACHEnvironment:
66     agent_selection = agents
67
68     def __init__(
69         self,
70         env_creator: callable,
71         parameter_generator: BaseParameterGenerator,
72         agents=dict(),
73         fill_random=True,
74         comm_schedule: Timeline=None,
75         seed=6234235,
76         TrajectoryClass = Trajectory,
77     ):
78         self.env_creator = env_creator
```

```

79         self.env = env_creator()
80         self.parameter_generator = parameter_generator
81
82         self.stored_agents = agents
83         self.fill_random = fill_random
84         self.seed = seed
85         self.TrajectoryClass = TrajectoryClass
86
87         if comm_schedule is None:
88             comm_schedule = Timelines(labels=["blackout", "checkins"])
89             comm_schedule.checkins.add_event(time=0, )
90         self.comm_schedule = comm_schedule
91
92         self.rendering=False
93
94         self.reset()
95
96         for k in agents.keys():
97             if not k in self.env.possible_agents:
98                 raise Exception(
99                     "Passed actor name is not in enviroments possible actors"
100                 )
101
102     def __deepcopy__(self, memo):
103         tmp_agents = {k: copy.deepcopy(a) for k, a in self.stored_agents.items()}
104
105         tmp = self.__class__(
106             self.env_creator,
107             self.parameter_generator,
108             tmp_agents,
109             copy.deepcopy(self.fill_random),
110             copy.deepcopy(self.comm_schedule),
111             copy.deepcopy(self.seed),
112         )
113
114         tmp.setstate(self.state)
115         return tmp
116
117     def _fillagents(self, fill_random=None):
118         if not fill_random:
119             fill_random = self.fill_random
120
121         self.action_spaces = {}
122         self.observation_spaces = {}
123
124         self.agents = dict()
125
126         for agt in self.env.possible_agents:
127             if agt in self.stored_agents.keys():
128                 self.agents[agt] = self.stored_agents[agt]
129             else:
130                 if fill_random:
131                     self.agents[agt] = RandomActor(agt)
132                 else:
133                     self.agents[agt] = DefaultActor(agt)
134
135             self.agents[agt].reset(self.env)
136             self.action_spaces[agt] = self.agents[agt].interface.action_dictionary
137             self.observation_spaces[agt] = self.agents[agt].interface.observation_space
138
139     def possible_models(self):
140         default = BasicActor, DefaultActor, RandomActor
141         return {role: default for role in self.env.possible_agents}
142
143     def set_coa(self, coas):
144         for agt, coa in coas.items():
145             self.agents[agt].update_coa(coa)
146
147     def get_coa(self):
148         return {role: agt.coa for role, agt in self.agents.items()}
149
150     def simulate(
151         self,
152         coas=None,
153         parameters=None,
154         agents=None,
155         state=None,
156         time_steps=None,
157         comm_steps=None,
158         render=False
159     ):

```

```

160         if comm_steps and time_steps:
161             logger.info(
162                 "Only one of `time_steps` and `comm_steps` can be defined at a time"
163             )
164             return
165
166         tmp_env = copy.deepcopy(self)
167         if agents is not None:
168             tmp_env.stored_agents = agents
169
170         if render:
171             tmp_env.start_rendering()
172
173         # Set new env params:
174         if parameters:
175             tmp_env.augment(parameters)
176
177         tmp_env.reset()
178         tmp_agts = tmp_env.agents
179
180         if state:
181             if parameters:
182                 logger.info(
183                     "Note: State parameters will overwrite passed parameters."
184                 )
185             tmp_env.setstate(state)
186
187         # Set new COA for agents
188         if coas:
189             for agt, coa in coas.items():
190                 tmp_agts[agt].update_coa(coa)
191
192         # Run simulation, recording trajectory
193         if time_steps:
194             tmp_env.step_env(steps=time_steps)
195
196         elif comm_steps:
197             for _ in range(comm_steps):
198                 tmp_env.step()
199
200         else:
201             ended = False
202             while not ended:
203                 tmp_env.step()
204                 terms = list(tmp_env.step_return[-1][2].values())
205                 trunks = list(tmp_env.step_return[-1][3].values())
206                 if all([a or b for a, b in zip(terms, trunks)]):
207                     ended = True
208
209         # Since trajectory starts at 0'th step
210         trajectory = tmp_env.state.trajectory
211
212         del tmp_env
213         del tmp_agts
214
215         return trajectory
216
217     def start_rendering(self, steps_per_frame = 10):
218         self.steps_per_frame = steps_per_frame
219         self.rendering=True
220
221     def stop_rendering(self):
222         self.rendering=False
223
224     def augment(self, parameters):
225         self.parameter_generator = parameters
226
227     def reset(self):
228         seed, parameters = self.parameter_generator.sample()
229         self.setup_env(seed, parameters)
230         self.state.cumulative_rews = {agt: 0 for agt in self.env.possible_agents}
231
232     def setup_env(self, seed, parameters):
233         self.env.augment(parameters)
234         self.env.seed(int(seed))
235         obs, info = self.env.reset()
236         self.step_return = [(obs, None, None, None, info)]
237
238         # Fill agents based on reset env
239         self._fillagents()
240

```

```

241     ## Reset agents, get their initial conditions.
242     self.agent_info = [
243         {role: agt.reset(env=self.env) for role, agt in self.agents.items()}
244     ]
245
246     initial_frame = None
247     if self.rendering:
248         initial_frame = self.env.render()
249
250     traj = self.TrajectoryClass(
251         self.env,
252         initial_return=self.step_return[0],
253         initial_frame=initial_frame
254     )
255
256     self.state = State(parameters, seed, traj, current_t=0)
257     self.alive = {agt:True for agt in self.agents.keys()}
258
259
260     def setstate(self, state: State):
261         self.setup_env(state.seed, state.parameters)
262
263         for step in state.trajectory[1:]:
264             step_return = self.env.step(step["action"])
265             self.step_return.append(step_return)
266
267             self.state.trajectory.add(
268                 env=self.env,
269                 action=step["action"],
270                 agent_info=step["agent_info"],
271                 step_return=step_return,
272             )
273
274         self.state.current_t = len(state.trajectory) - 1
275
276
277     def last(self):
278         return self.step_return[-1]
279
280     def step(self, coas=None):
281         ## This is the step through the coa
282         if coas:
283             self.set_coa(coas)
284
285         next_comm, _ = self.comm_schedule.checkins.next_event(self.state.current_t)
286         if next_comm:
287             next_comm = next_comm - self.state.current_t
288
289         logger.debug("current_t: %s, next_com: %s", self.state.current_t, next_comm) # DEBUG
290         self.step_env(steps=next_comm)
291
292         return self.step_return[-1], self.step_end
293
294     def step_env(self, steps=1, coas=None):
295         self.step_end = {
296             "coa_done": [],
297             "term_or_trunc": False,
298             "steps_reached": False
299         }
300
301         if coas:
302             self.set_coa(coas)
303
304         t = self.state.current_t-1
305         running = True
306
307         while running:
308             t += 1
309             logger.debug("Env Step: %s", t) # DEBUG
310
311             action = dict()
312             agent_info = dict()
313
314             for role, agt in self.agents.items():
315                 if self.alive[role]:
316                     action[role], agent_info[role] = agt.get_action(
317                         self.step_return[-1][0][role], t
318                     )
319                 else:
320                     action[role] = None
321                     agent_info[role] = "not_alive"

```

```

322         logger.debug("Env Action: %s", action)      # DEBUG
323
324     obs, rwds, terms, trunks, info = self.env.step(action)
325
326     frame = None
327     if self.rendering:
328         if t % self.steps_per_frame == 0:
329             frame = self.env.render()
330
331     returns = (
332         {role: self.agents[role].process_observations(o, t) for role, o in obs.items()},
333         rwds, terms, trunks, info
334     )
335
336     for role, rwd in returns[1].items():
337         self.state.cummulative_rews[role] += rwd
338
339     self.agent_info.append(agent_info)
340     self.step_return.append(returns)
341     self.state.trajectory.add(
342         env=self.env,
343         action=action,
344         step_return=returns,
345         agent_info=agent_info,
346         frame=frame)
347
348     self.alive = {agt: not (returns[2][agt] or returns[3][agt]) for agt in self.agents.keys()}
349
350     break_for_new_COA = False
351
352     if self.comm_schedule.ALLOW_AGENT_BREAK:
353         for role in self.agents.keys():
354             if self.comm_schedule.blackouts.get(t, role) is not None:
355                 if self.agent_info[-1][role]["coa_done"]:
356                     self.step_end["coa_done"].append(role)
357                     break_for_new_COA = True
358                     running = False
359
360     if not any(self.alive.values()):
361         self.step_end["term_or_trunc"] = True
362         running = False
363
364     if steps:
365         if t >= (self.state.current_t + steps):
366             running = False
367
368     if steps:
369         if t == self.state.current_t + steps - 1:
370             # We reached the end, even if other things would have terminated it
371             self.step_end["steps_reached"] = True
372
373     logger.debug("Env step end: %s", self.step_end)  # DEBUG
374     self.state.current_t = t + 1
375
376     #####
377     # COA Level Interface:
378     #####
379
380     def get_traj_from_coas(self, coas, params=None, from_state=None, render=False):
381         traj = self.simulate(
382             coas,
383             parameters=params,
384             state=from_state,
385             render=render
386         )
387         return traj
388
389     def get_traj_from_plan(self,
390         plan,
391         params=None,
392         from_state=None,
393         render=False
394     ):
395         coas = {role: coa for role, coa in plan.items()}
396
397         agent_dict = dict()
398         for role, model_class in plan.model_classes.items():
399             agent_dict[role] = model_class(role, **plan.model_params[role])
400
401         traj = self.simulate(

```

```

403         coas,
404         parameters=params,
405         agents=agent_dict,
406         state=from_state,
407         render=render
408     )
409
410     return traj
411
412 def trajectory_telemetry(
413     self,
414     traj,
415     components=None,
416     labels=None,
417     players=None,
418     ao="observations",
419     plan=None,
420     env=None,
421     cmap=None
422 ):
423
424     if players is None:
425         players = list(traj.step_returns[0][0].keys())
426
427     if cmap is None:
428         default_cycle = plt.rcParams['axes.prop_cycle'].by_key()['color']
429         cmap = lambda i: default_cycle[i%len(default_cycle)]
430
431     traj_len = len(traj)
432
433     rewards = Telemetry(
434         name="Reward",
435         title="Reward",
436         xscale = np.array(range(traj_len)),
437         xlabel = "Time Step",
438         ylabel = "Reward",
439     )
440
441     rewards.set_data(np.zeros([len(players),traj_len]))
442
443     for i, p in enumerate(players):
444         rewards.data_labels[i] = p
445         rewards.colors[i] = cmap(i)
446
447         for j, t in enumerate(traj.step_returns[1:]): # No initial reward
448             rewards.data[i, j] = t[1][p]
449
450     return [rewards]
451
452 # Note: This should be able to handle plotting
453 # without having to have the original env spun up.
454 def plot_trajectory_component(
455     self,
456     traj,
457     components=[0],
458     labels=None,
459     players=None,
460     ao="observations",
461     plan=None,
462     env=None,
463     render_mode="rgb_array" # Union["rgb_array", "matplotlib", "plotly"]
464 ):
465
466     if not players:
467         players = list(traj.step_returns[0][0].keys())
468
469     if not type(players) is list:
470         players = [players]
471
472     create_labels = False
473     if not labels:
474         labels = dict()
475         create_labels = True
476
477     if ao == "observations":
478         data = {}
479         all_components = False
480         if not components:
481             all_components = True
482
483     for p in players:

```

```

484         action_list = []
485         for t in traj.step_returns:
486             t = t[0]
487             if all_components:
488                 components = list(range(len(np.array(t[p]).reshape(-1))))
489
490             action_list.append(t[p][components])
491
492         data[p] = np.array(action_list)
493         if create_labels:
494             labels[p] = [f"{p}: {c}" for c in components]
495
496 if ao == "actions":
497     data = {}
498     all_components = False
499     if not components:
500         all_components = True
501
502     for p in players:
503         action_list = []
504         for t in traj.actions[1:]:
505             if all_components:
506                 components = list(range(len(np.array(t[p]).reshape(-1))))
507
508             action_list.append(t[p][components])
509
510         data[p] = np.array(action_list)
511         if create_labels:
512             labels[p] = [f"{p}: {c}" for c in components]
513
514 if ao == "rewards":
515     components = [0]
516     data = traj.step_returns
517     xs = np.zeros([len(players), len(components), len(data)-1])
518
519     for j, t in enumerate(data):
520         if j > 0: # The initial state has no reward
521             for i, p in enumerate(players):
522                 xs[i, :, j] = t[1][p]
523
524 n_axes = sum([s.shape[1] for s in data.values()])
525 # Setup Figures
526 w = n_axes // 2
527 w_m = w + n_axes % 2
528
529 f = plt.gcf()
530 axes = []
531 for i in range(w):
532     for j in range(2):
533         axes.append(plt.subplot2grid((w_m, 4), (i, 2 * j), 1, 2))
534
535 if n_axes % 2:
536     axes.append(plt.subplot2grid((w + 1, 4), (w, 1), 1, 2))
537
538 # Get Data
539
540 idx = 0
541 for p in players:
542     for j in range(data[p].shape[1]):
543         axes[idx].plot(data[p][:, j])
544         axes[idx].title.set_text(labels[p][j])
545         idx += 1
546
547 # 15 = axes[0].legend(
548 #     bbox_to_anchor=(0.5, -0.05), loc="lower center", bbox_transform=f.transFigure
549 # )
550
551 plt.subplots_adjust(left=0.1, right=0.9, hspace=0.3, wspace=0.5)
552
553 fig = plt.gcf()
554 if render_mode == "matplotlib":
555     return fig
556
557 canvas = FigureCanvas(fig)
558 canvas.draw()
559
560 data = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
561 data = data.reshape(fig.canvas.get_width_height()[::-1] + (3,))
562 plt.close(fig)

```



```

565
566     if render_mode == "rgb_array":
567         return data
568
569     if render_mode == "plotly":
570         import plotly.express as px
571         fig = px.imshow(data)
572         return fig
573
574
575
576
577
578 #####
579 ## Example Usage
580 #####
581
582 if __name__ == "__main__":
583     from examples.MAInspection.env import MultiInspect
584
585     ## Environmental Parameters
586     env_params = {"_OBS_REWARD":.01, "num_deputes": 3}
587
588     # Actor Parameters
589     COACH_params = {
590         "Agents": {
591             "player_0": {
592                 "class_name": "DefaultActor"
593                 # params: {"parameter": "value"} # If the agent has setup parameters
594             },
595             "player_1": {"class_name": "RandomActor"},
596             "player_2": {"class_name": "DefaultActor"},
597         }
598     }
599
600     # Create actors from parameters
601     agent_dict = dict()
602     if "Agents" in COACH_params.keys():
603         for role, agent in COACH_params["Agents"].items():
604             agent_class = COACHEnvironment.agent_selection.Agents[agent["class_name"]]
605             agent_dict[role] = agent_class(role, **agent.get("params", dict()))
606
607     # Example communication scheudle.
608     comms = CommunicationSchedule.repeating(checkin_frequency=10)
609
610     # Wrap parameters in parameter generator
611     parameter_generator = SeededParameterGenerator(23423, env_params)
612
613     env = COACHEnvironment(
614         env_creator = MultiInspect,
615         default_parameters = parameter_generator,
616         comm_schedule = comms,
617         fill_random = True,
618         agents = agent_dict
619     )
620
621     for agt in env.agents.values():
622         print(agt.interface)
623         input()
624
625     for i in range(10):
626         print(env.step())
627         for role, actions in env.action_spaces.items():
628             print(f"{role}\t {actions}")
629         input("Press Enter For Next Observation")

```

1.4 directors.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from env import COACH_PettingZoo
16 from stable_baselines3 import PPO
17
18 import copy
19
20 import logging
21 logger = logging.getLogger(__name__)
22
23
24 class SB3_PPO_Director:
25     def __init__(self, env_creator, COACHEnvClass, params, model_path):
26         self.env_creator = env_creator
27         self.COACHEnvClass = COACHEnvClass
28         self.params = params
29
30         self.env = COACH_PettingZoo(env_creator=env_creator, COACHEnvClass=COACHEnvClass)
31         self.env.augment(params)
32
33         if model_path.endswith(".zip"):
34             model_path = model_path[:-4]
35
36         self.policy = PPO.load(model_path)
37
38     def generate_coas(self, params=None):
39         if not params:
40             params = self.params
41
42         self.env.augment(params)
43
44         obs, info = self.env.reset()
45         running = True
46
47         while running:
48             act = {"director": self.policy.predict(obs["director"], deterministic=False)[0]}
49
50             # logger.debug("Obs From Director: %s", obs["director"]) # DEBUG
51             # logger.debug("Action From Director: %s", act) # DEBUG
52
53             obs, reward, term, trunc, info = self.env.step(act)
54
55             if all([a or b for a,b in zip(term.values(), trunc.values())]):
56                 running = False
57
58         coas = dict()
59         for role, agent in self.env.coa_env.agents.items():
60             coas[role] = agent.coa
61
62         traj = copy.deepcopy(self.env.coa_env.state.trajectory)
63         return coas, traj
```

1.5 env.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import sys
16
17 from gymnasium.spaces import Box
18 from pettingzoo.utils.env import ParallelEnv
19 import gymnasium as gym
20 from typing import Any
21 import numpy as np
22 import argparse
23
24 from coach import (
25     SeededParameterGenerator,
26     BaseParameterGenerator,
27 )
28
29 from utilities.planning import COA
30
31 from coach import COACHEnvironment, CommunicationSchedule
32
33 import logging
34 logger = logging.getLogger(__name__)
35
36
37 def get_env_class(args: argparse.Namespace):
38     """Returns the class to use, based on input arguments
39
40     Parameters
41     -----
42     args: argparse.Namespace
43         arguments that were passed to the `main()` function
44
45     Returns
46     -----
47     class
48         the class to use in creating env objects
49     """
50     return COACH_PettingZoo
51
52
53 class COACH_PettingZoo(gym.Wrapper, ParallelEnv):
54     def __init__(self, env_creator, COACHEnvClass):
55         self.env_creator = env_creator
56         self.COACHEnvClass = COACHEnvClass
57         self.fake_render_mode = "rgb_array"
58
59     def _setup(self, params):
60         self.current_params = params
61         self.COACH_params = COACH_params = params["COACH_params"]
62         self.env_params = env_params = params["env_params"]
63
64         # Set up parameter generator
65         if COACH_params["stochastic"]:
66             self.parameter_generator = SeededParameterGenerator(23423, env_params)
67         else:
68             self.parameter_generator = BaseParameterGenerator(23423, env_params)
69
70         # Set up comm schedule
71         if "FIXED_STEPS_PER_COM" in COACH_params.keys():
72             schedule_param = COACH_params["FIXED_STEPS_PER_COM"]
73             self.comm_schedule = CommunicationSchedule.repeating(**schedule_param)
74         else:
75             logger.info("Communication Schedule Is Non-repeating")
76             self.comm_schedule = CommunicationSchedule(length=0)
77
78         self.ACTION_PADDING = COACH_params.get("ACTION_PADDING", 0)
```

```

79     self.MIN_NEXT_ACTION_TIME = COACH_params.get("MIN_NEXT_ACTION_TIME", 1)
80     self.MAX_NEXT_ACTION_TIME = COACH_params.get("MAX_NEXT_ACTION_TIME", np.inf)
81
82     # Setup Agents
83     agent_dict = dict()
84
85     if "Agents" in self.COACH_params.keys():
86         for role, agent in self.COACH_params["Agents"].items():
87             logger.info(f"##### {role} {agent}")
88             agent_class = self.COACHEnvClass.agent_selection.Agents[agent["class_name"]]
89             agent_dict[role] = agent_class(role, **agent.get("params", dict()))
90
91     # Create COA Env
92     self.coa_env = self.COACHEnvClass(
93         env_creator=self.env_creator,
94         parameter_generator=self.parameter_generator,
95         agents=agent_dict,
96         fill_random=False,
97         comm_schedule=self.comm_schedule,
98         seed=COACH_params["seed"],
99     )
100
101     self.coa_env.augment(self.parameter_generator)
102     self.coa_env.reset()
103
104     self.stochastic = COACH_params["stochastic"]
105
106     self.players = list(self.coa_env.agents.keys())
107     self.player_actions = self.coa_env.action_spaces
108
109     self.possible_agents = ["director"]
110     self.agents = ["director"]
111
112     # Action spaces
113     self.player_actions = dict()
114     all_lows = []
115     all_highs = []
116     self.action_indexes = []
117     idx = 0
118     for role in self.players: # Preserve order. Dicts should do this now but just to be safe
119         # Process actions
120         actions = self.coa_env.action_spaces[role]
121         lows = []
122         highs = []
123         self.action_indexes
124         for label, action in actions.items():
125             # Need to add an initial entry for the start time
126             lows.append(np.concatenate([[self.MIN_NEXT_ACTION_TIME],
127                                         np.array(action.low).reshape(-1)]))
128             highs.append(np.concatenate([[self.MAX_NEXT_ACTION_TIME],
129                                         np.array(action.high).reshape(-1)]))
130
131             L = len(np.array(action.low).reshape(-1)) + 1
132             self.action_indexes.append((role, label, idx, idx + L))
133             idx = idx + L
134
135         lows = np.concatenate(lows)
136         highs = np.concatenate(highs)
137         all_lows.append(lows)
138         all_highs.append(highs)
139         self.player_actions[role] = Box(low = lows, high = highs)
140
141     self.action_spaces = {"director": Box(
142         low = np.concatenate(all_lows),
143         high = np.concatenate(all_highs)
144     )}
145
146     # Observation spaces
147     self.player_observations = dict()
148     all_lows = []
149     all_highs = []
150     for role in self.players: # Preserve order. Dicts should do this now but just to be safe
151         # Process actions
152         observations = self.coa_env.observation_spaces[role]
153         low = np.array(observations.low).reshape(-1)
154         high = np.array(observations.high).reshape(-1)
155
156         all_lows.append(low)
157         all_highs.append(high)
158         self.player_observations[role] = observations

```

```

158
159     self.observation_spaces = {"director": Box(
160         low = np.concatenate(all_lows),
161         high = np.concatenate(all_highs)
162     )
163 }
164
165 #####
166 # Standard PettingZoo Interface Functions
167 #####
168 def observation_space(self, role):
169     return self.observation_spaces[role]
170
171 def action_space(self, role):
172     return self.action_spaces[role]
173
174 def reset(self, seed=0, options=None):
175     self.coa_env.reset()
176     self.steps = 0
177     self.cumulative_rew = np.zeros(len(self.possible_agents))
178
179     self.coas = dict()
180     for role in self.players:
181         self.coas[role] = COA()
182
183     return (
184         {"director": self._process_observations(self.coa_env.last())},
185         {},
186     )
187
188 def augment(self, params):
189     self._setup(params)
190
191 def seed(self, seed):
192     self.parameter_generator.setseed(seed)
193
194 def render(self, components=None, ao="rewards"):
195     return self.coa_env.plot_trajectory_component(
196         self.coa_env.state.trajectory,
197         components = components,
198         ao = ao
199     )
200
201 def step(self, action, render=False):
202     self._process_actions(action)
203     logger.debug("Director Step: COA: %s", self.coas)
204     last_returns, step_end = self.coa_env.step(self.coas)
205     logger.debug("Time: %s, Step End: %s", self.coa_env.state.current_t, step_end)
206
207     # Process terminations and truncations
208     term = False
209     trunc = False
210
211     if step_end["term_or_trunc"]:
212         if all(list(last_returns[2].values())):
213             # Unless everyone terminates, somebody must have truncated.
214             term = True
215         else:
216             trunc = True
217
218     # Process reward
219     reward_til_now = sum(self.coa_env.state.cumulative_rews.values())
220     step_reward = reward_til_now - self.cumulative_rew
221     self.cumulative_rew = reward_til_now
222
223     return (
224         {"director": self._process_observations(last_returns)},
225         {"director": step_reward}, ## Reward
226         {"director": term}, ## Term
227         {"director": trunc}, ## Trunc
228         {"director": {}}, ## Info
229     )
230
231 #####
232 # Conversion between actions and COAs
233 #####
234
235 def _process_actions(self, action):
236     action = action["director"]
237
238     coas = {role:[] for role in self.players}

```

```

239
240     logger.debug("Processing Action to COA")
241     for role, label, i0, i1 in self.action_indexes:
242         logger.debug("%s, %s, %s, %s, %s, %s", role, label, i0, i1, np.floor(action[i0]) +
243             self.coa_env.state.current_t, action[i0+1:i1])
244         event = {
245             "start": np.floor(action[i0]) + self.coa_env.state.current_t,
246             "label": label,
247             "parameters": action[i0+1:i1],
248             "role": role,
249         }
250         coas[role].append(event)
251
252     for role in self.players:
253         self.coas[role].add_events_from_dict(coas[role])
254
255     def _process_observations(self, last_returns):
256         # When working with a specific env you almost certainly want to change this
257         # as there may be a ton of redundant information in the combined observation
258         # space
259         obs = np.concatenate([last_returns[0][role].reshape(-1) for role in self.players])
260         return obs
261
262     #####
263     # Wrapper Functions
264     #####
265
266     def __getattr__(self, name: str) -> Any:
267         """Returns an attribute with ``name``, unless ``name`` starts with an underscore."""
268
269         if name == "coa_env":
270             if "coa_env" not in self.__dict__.keys():
271                 self.__dict__["coa_env"] = None
272                 return self.__dict__["coa_env"]
273
274         if name in self.__dict__:
275             return self.__dict__[name]
276
277         if name == "unwrapped":
278             return self.coa_env.env
279
280         if name == "parallel_env":
281             return self
282
283         return getattr(self.coa_env.env, name)
284
285     #####
286     # DASH Viewer Functions
287     #####
288
289     def set_fake_render_mode(self, fake_render_mode):
290         self.fake_render_mode = fake_render_mode
291
292     # %%
293     if __name__ == "__main__":
294         params = {
295             "COACH_params": {
296                 "stochastic": True,
297                 # "FIXED_STEPS_PER_COM": {
298                 #     "checkin_frequency": 10
299                 # },
300                 "ACTION_PADDING": 0,
301                 "MIN_NEXT_ACTION_TIME": 1,
302                 "MAX_NEXT_ACTION_TIME": 10,
303                 "Agents": {
304                     "pursuer_0": {
305                         "class_name": "DefaultActor",
306                         "params": {"max_action_len": 6}
307                     },
308                     "pursuer_1": {
309                         "class_name": "DefaultActor",
310                         "params": {"max_action_len": 5}
311                     },
312                 },
313                 "seed": 453413,
314             },
315             "env_params": {"n_pursuers": 2}
316         }
317
318     env = COACH_PettingZoo(env_creator=PettingZooEnv, COACHEnvClass=COACHEnvironment)

```

```

319
320     env.augment(params)
321     env.reset()
322
323     print("#*20, "COA Gym Information", "#*20)
324     print("Players:", env.players)
325     print("Observation Space:", env.observation_spaces["director"].shape)
326     print("Action Space:", env.action_spaces["director"].shape)
327     print("Sample action:", env.action_spaces["director"].sample())
328
329     for i in range(50):
330         print("Turn", i)
331         obs,rew,term,trunc,info = env.step({"director": env.action_spaces["director"].sample()})
332         if all([a or b for a,b in zip(term.values(), trunc.values())]):
333             print("Environment has terminated.")
334             break
335
336     env.render(ao="actions")
337     # %%

```

1.6 params.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from typing import Any, Type
16
17 def get_env_param_class(args: Any) -> Type:
18     """Returns the class to use, based on input arguments
19
20     Parameters
21     -----
22     args: argparse.Namespace
23         arguments that were passed to the `main()` function
24
25     Returns
26     -----
27     class
28         the class to use in creating env parameter objects
29     """
30     return EnvParams
31
32
33 """Defines base class for environment parameter classes."""
34
35 from typing import Any, Optional
36
37 class EnvParams:
38     """Base class for environment params."""
39
40     def __init__(self, args: Any, param_section_name: str = "env_params") -> None:
41         self.args = args
42         self._params = getattr(args, param_section_name, {})
43
44     def __getitem__(self, key: str) -> Any:
45         """Return value stored for key from the params dict."""
46         return self._params[key]
47
48     def __setitem__(self, key: str, value: Any) -> None:
49         """Set the value for key in the params dict."""
50         self._params[key] = value
51
52     def get(self, key: str, default: Optional[Any] = None) -> Any:
53         """Return value for key from the params dict, or None if it doesn't exist."""
54         try:
55             return self._params[key]
56         except KeyError:
57             return default
58
59     def get_mutated_params(self) -> "EnvParams":
60         """Return a mutated copy of the params"""
61         raise NotImplementedError(
62             f"get_mutated_params has not been implemented in {type(self)}"
63         )
64
65     def checkpoint(self, folder: str) -> None:
66         """Save a checkpoint in the given folder."""
67         raise NotImplementedError(
68             f"checkpoint has not been implemented in {type(self)}"
69         )
70
71     def reload(self, folder: str) -> None:
72         """Read a checkpoint from the given folder."""
73         raise NotImplementedError(f"reload has not been implemented in {type(self)}")
74
```


1.7 requirements.txt

```
1 astropy==5.3.4
2 dash==2.15.0
3 gymnasium==0.29.1
4 imageio==2.34.0
5 matplotlib==3.8.2
6 networkx==3.2.1
7 numpy==1.26.4
8 pandas==2.2.0
9 pettingzoo==1.24.3
10 Pillow==10.2.0
11 plotly==5.18.0
12 poliastro==0.17.0
13 PyYAML==6.0.1
14 PyYAML==6.0.1
15 scipy==1.12.0
16 stable_baselines3==2.2.1
17 SuperSuit==3.9.2
18 tensorflow==2.15.0.post1
19 torch==2.2.0
20 tqdm==4.66.2
```

1.8 texify.py

```
1  #!/usr/bin/env python3
2
3  import os, re, argparse
4
5  latex_header = r'''
6  \documentclass{article}
7  \usepackage[margin=1in]{geometry}
8  \usepackage{textcomp}
9  \usepackage{listingsutf8}
10 \usepackage{hyperref}
11 \usepackage[dvipsnames]{xcolor}
12 \definecolor{darkgreen}{rgb}{0,0.5,0}
13 \definecolor{lightblue}{rgb}{0.2,0.5,1}
14 \hypersetup{colorlinks=true, linkcolor=blue}
15 \lstset{
16     numbers=left,
17     upquote=true,
18     breaklines=true,
19     tabsize=4,
20     showstringspaces=false,
21     showspace=false,
22     breakatwhitespace=true,
23     <SYNTAX_HIGHLIGHTING>
24 }
25 \begin{document}
26 \tableofcontents
27 \newpage
28 '''
29
30 styles = {
31     'default': r'''
32         basicstyle=\ttfamily\scriptsize,
33         keywordstyle=\ttfamily,
34         commentstyle=\ttfamily\color{darkgreen},
35         stringstyle=\ttfamily\color{blue},
36     ''',
37     'dark': r'''
38         backgroundcolor=\ttfamily\color{black},
39         basicstyle=\ttfamily\color{white}\scriptsize,
40         keywordstyle=\ttfamily,
41         commentstyle=\ttfamily\color{green},
42         stringstyle=\ttfamily\color{lightblue},
43     ''' # xterm-mode
44 }
45
46 # Governs syntax highlighting
47 file_types = {
48     '.py': 'Python',
49     '.c': 'C',
50     '.d': 'C',
51     '.m': 'Matlab',
52     '.r': 'R',
53     '.sh': 'bash',
54     '.bash': 'bash',
55     '.cpp': 'C++',
56     '.cc': 'C++',
57     '.pl': 'Perl',
58     '.tex': 'TeX',
59     '.f': 'Fortran',
60     '.for': 'Fortran',
61     '.ftn': 'Fortran',
62     '.f90': 'Fortran',
63     '.f95': 'Fortran',
64     '.f03': 'Fortran',
65     '.f08': 'Fortran',
66     '.csh': 'csh',
67     '.ksh': 'ksh',
68     '.lisp': 'lisp',
69     '.lsp': 'lisp',
70     '.cl': 'lisp',
71     '.l': 'lisp',
72     '.scm': 'lisp',
73     '.go': 'Go',
74     '.hs': 'Haskell',
75     '.lhs': 'Haskell',
76     '.bat': 'command.com',
77     '.awk': 'Awk',
78 }
```

```

79
80 def main() -> None:
81     parser = argparse.ArgumentParser(usage='%s [-d DIR] [-i extension ...]\n' % __file__
82         + 'example: %s -d ./src -i foo.m -i makefile .c .d .py\n\n' % __file__,
83         description='Will search under DIR for all source files with the specified file extensions, and
            compile them into a LaTeX file.')
84     parser.add_argument('--dir', '-d', help='root directory under which to search', default='.')
85     parser.add_argument('--include', '-i', action='append', help="Explicitly include a file even if it
            doesn't match the extension list", default=[])
86     parser.add_argument('--style', default='default', choices=styles.keys(), help='Changes syntax
            highlighting, etc.')
87     parser.add_argument('extension', nargs='+', help='Only files with these extensions will be included
            (leading dot optional)')
88     args = parser.parse_args()
89
90     # Permit valid extensions to be input with or without the dot
91     args.extension = [a if ('.' == a[0]) else '%s' % a
92         for a in args.extension]
93
94     # Make relative to base path, escape underscores
95     def format_path(path: str) -> str:
96         if path == args.dir: return '/'
97         assert (path[0:len(args.dir)+1] == args.dir + '/') or (path[0:len(args.dir)+1] == args.dir +
            '\\')
98         return re.sub('_', r'\_', path[len(args.dir)+1:])
99
100     # Print single file
101     def dumpsrc(dirpath: str, fname: str) -> str:
102         path = '%s/%s' % (dirpath, fname)
103         escaped = format_path(path)
104         print(r'\subsection[%s]{%s}' % (os.path.basename(escaped), escaped))
105         ext = os.path.splitext(f)[1]
106         if ext in file_types:
107             s = r'\lstinputlisting[language=%s]{%s}' % (file_types[ext], path)
108         else:
109             s = r'\lstinputlisting{%s}' % path
110         return '%s\n%s\n' % (s, r'\newpage')
111
112     def print_header() -> None:
113         s = latex_header.replace(r'<SYNTAX_HIGHLIGHTING>', styles[args.style].strip(), 1)
114         print(s.strip())
115
116     print_header()
117
118     dirs = {dirpath:fnames for dirpath, _, fnames in os.walk(args.dir)}
119     includes = {os.path.realpath(f):f for f in args.include}
120     for dirpath in sorted(dirs):
121         fnames = dirs[dirpath]
122         src = sorted([f for f in fnames
123             if (os.path.splitext(f)[1] in args.extension) or (os.path.realpath(f) in includes)])
124         if 0 == len(src): continue
125
126         print(r'\section[%s]' % format_path(dirpath))
127         for f in src:
128             print(dumpsrc(dirpath, f))
129
130         # Don't include files twice just because they're explicitly included with -i
131         f = os.path.realpath(f)
132         if f in includes:
133             del includes[f]
134
135     # Any explicitly included files that weren't already covered (i.e. those outside args.path)
136     if len(includes):
137         print(r'\section{Miscellaneous}')
138         for _, f in includes.items():
139             f = args.dir + '/' + os.path.relpath(f, args.dir)
140             print(dumpsrc(os.path.dirname(f), os.path.basename(f)))
141
142     print(r'\end{document}')
143
144     main()

```

2 DASH

2.1 DASH/app.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import os
16 import re
17 import sys
18
19
20 # Utility Libraries
21 import numpy as np
22 import copy
23 import json
24
25 # Import Dash Things
26 from dash import Dash, html, dcc, Input, Output, callback, State, ALL, no_update, ctx, MATCH
27 import plotly.express as px
28 import plotly.graph_objects as go
29
30 # Import Custom Things
31 from DASH.html_objects import app_layout, plan_menu, actions_display
32 from DASH.dash_utilities import callback_tools as ct
33
34 # Import Env Things:
35 from DASH.coach_integration import COACHIntegration
36 from coach import COACHEnvironment
37
38 # Logging
39 import logging
40
41 logging.basicConfig(level=logging.DEBUG, format="%(levelname)s:%(name)s:%(message)s")
42 logger = logging.getLogger(__name__)
43
44 pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
45                   name.startswith("pymunk")]
46
47 for log_handler in pymunk_loggers:
48     log_handler.setLevel(logging.INFO)
49
50 #####
51 # Inputs
52 #####
53
54 ### Plan Selector
55 @callback(
56     Output("update_backend_coa", "data", allow_duplicate=True),
57     Input({"type": "plan", "id": ALL}, "n_clicks"),
58     prevent_initial_call=True,
59 )
60
61 def select_plan(plan):
62     logger.debug("INPUT: select_plan")
63
64     plan_id = ctx.triggered_id['id']
65     logger.debug("\t %s %s %s", "select_plan", plan, plan_id)
66     if plan[plan_id] > 0:
67         ct.env_factory.plans.set_current_plan(plan_id)
68
69     return True
70
71
72
73 ### Plan View Checkboxes
74 @callback(
```

```

75     Output("update_backend_coa", "data", allow_duplicate=True),
76     Input({"type": "plan_view", "id": ALL}, "value"),
77     prevent_initial_call=True,
78 )
79 def view_plan(plan_checks):
80     logger.debug("INPUT: view_plan")
81
82     plan_id = ctx.triggered_id['id']
83     logger.debug("\t %s %s %s", "view_plan", plan_checks, plan_id)
84     checked = len(plan_checks[plan_id])>0
85
86     if checked:
87         ct.env_factory.plans.active.add(plan_id)
88         logger.debug("\t %s %s", "plan added, active plans:", ct.env_factory.plans.active)
89     else:
90         if plan_id in env_factory.plans.active:
91             env_factory.plans.active.remove(plan_id)
92             logger.debug("\t %s %s", "plan removed, active plans:", ct.env_factory.plans.active)
93         else:
94             return no_update
95
96     return True
97
98 ### Interface Selector
99 @callback(
100     Output("update_backend_coa", "data", allow_duplicate=True),
101     Input({"type": "agent_interface", "role": ALL}, "value"),
102     prevent_initial_call=True,
103 )
104 def choose_interface(agent_interface):
105     logger.debug("INPUT: choose_interface")
106
107     # Get the role that changed.
108     role = ctx.triggered_id['role']
109     interface = ct.get_role_from_callback(role, agent_interface)
110     logger.debug("\t %s %s", "choose_interface:", role)
111
112     logger.debug("\t %s %s %s %s %s %s", "Interface Selector:", role, "new interface:", interface, "old
113         interface:", ct.env_factory.plans.current.get_interface(role)[0])
114     if interface == ct.env_factory.plans.current.get_interface(role)[0]:
115         logger.debug("INPUT: choose_interface - no_update")
116         return no_update
117     else:
118         logger.debug("INPUT: choose_interface - updated")
119         ct.env_factory.plans.current.set_interface(role, interface)
120         return True
121
122
123 ### Model Selector
124 @callback(
125     Output("update_backend_coa", "data", allow_duplicate=True),
126     Input({"type": "onboard_model", "role": ALL}, "value"),
127     prevent_initial_call=True,
128 )
129 def select_model(onboard_model):
130     logger.debug("INPUT: select_model")
131     role = ctx.triggered_id['role']
132
133     model = ct.get_role_from_callback(role, onboard_model)
134     if model == ct.env_factory.plans.current.get_model(role):
135         return no_update
136     else:
137         logger.debug("\t Setting Model for Plan %s %s %s %s %s", ct.env_factory.plans.current.id, "for
138             role", role, "to", model)
139         ct.env_factory.plans.current.set_model(role, model)
140         return True
141
142
143 ### Action Parameters
144 @callback(
145     Output("update_backend_coa", "data", allow_duplicate=True),
146     Input({"type": "actionparam", "plan": ALL, "time": ALL, "type": ALL, "index": ALL, "role": ALL},
147         "value"),
148     State({"type": "actionparam", "plan": ALL, "time": ALL, "type": ALL, "index": ALL, "role": ALL},
149         "id"),
150     prevent_initial_call=True,
151 )
152 def change_action_params(params, id):
153     logger.debug("INPUT: change_action_params")

```

```

152     logger.debug("\t %s %s", "calling id:", id)
153
154     if ctx.triggered_id is None:
155         ## This got called by a plan change
156         return no_update
157
158     logger.debug("\t %s %s", "ctx.triggered_id:", ctx.triggered_id)
159
160     role = ctx.triggered_id['role']
161     logger.debug("\t %s %s %s %s", "New Params:", params, "role:", role)
162
163     locks["action_card"] = True
164     logger.debug("\t %s", "LOCKING ACTION PARAMS")
165
166     i = id.index(ctx.triggered_id)
167     plan = ct.env_factory.plans.get(ctx.triggered_id["plan"])
168     coa = plan.coas[ctx.triggered_id["role"]]
169     event = coa.get(time = ctx.triggered_id["time"], label = ctx.triggered_id["type"])
170
171     logger.debug("\t %s %s", "Param_Update: Current COA" , coa)
172
173     event.parameters[ctx.triggered_id["index"]] = params[i]
174
175     logger.debug("\t %s %s", "Param_Update: New COA" , coa)
176
177     return True
178
179     ### Timeline Selector
180
181     @callback(
182         Output("update_backend_coa", "data", allow_duplicate=True),
183         Input({"type": "timeline", "role": ALL}, "value"),
184         prevent_initial_call=True,
185     )
186
187     def edit_timeline(new_timelines):
188         role = ctx.triggered_id['role']
189
190         logger.debug("\t %s", "INPUT: edit_timeline")
191         logger.debug("\t %s %s", "new_timelines", new_timelines)
192         logger.debug("\t %s %s", "Old COA:", ct.env_factory.plans.current.coas[role])
193
194         # Figure out which timeline event changed
195         changed_timeline = ct.get_role_from_callback(role, new_timelines)
196         logger.debug("\t %s %s %s", "updated timeline:", role, changed_timeline)
197
198         if len(changed_timeline)==0:
199             logger.debug("INPUT: edit_timeline - no_update")
200             return no_update
201
202         old_timeline = ct.env_factory.plans.current.get_dash_timelines()[role]
203
204         # If we drag a timepoint across another it will change it's position in the ordering
205         # so we need some extra logic around that
206         old = []
207         new = []
208
209         for j_old, j_new in zip(old_timeline, changed_timeline):
210             if not j_old == j_new:
211                 old.append(j_old)
212                 new.append(j_new)
213
214         if len(old)==0:
215             logger.debug("INPUT: edit_timeline - no_update")
216             return no_update
217
218         # Check if we've moved one point past another
219         if len(old)>1:
220             if new[0] == old[1]:
221                 j_new = new[1]
222                 j_old = old[0]
223             else:
224                 j_new = new[0]
225                 j_old = old[1]
226         else:
227             j_new = new[0]
228             j_old = old[0]
229
230         # Check if we're moving one point on top of another
231         if j_new in old_timeline:
232             # Kind of dumb trick to make it bounce to the side you're dragging it from.

```

```

233         sign = -(j_new - j_old)/abs(j_new - j_old)
234         while j_new in changed_timeline:
235             j_new += sign*1
236
237         logger.debug("\t %s %s", "New Timelines:", changed_timeline)
238         logger.debug("\t %s %s", "Old Timelines:", old_timeline)
239         logger.debug("\t %s %s", "Change Index:", j_old, j_new)
240
241         ct.env_factory.plans.current.coas[role].move_events(time=j_old, to=j_new)
242
243         logger.debug("\t %s %s", "New COA:", ct.env_factory.plans.current.coas[role])
244
245         logger.debug("INPUT: edit_timeline - Update Backend")
246         return True
247
248 #####
249 # Buttons
250 #####
251
252 ### New Plan
253 @callback(
254     Output("update_backend_coa", "data", allow_duplicate=True),
255     Input("button_new_plan", "n_clicks"),
256     prevent_initial_call=True,
257 )
258 def new_plan(nclicks):
259     logger.debug("BUTTON: new_plan")
260     ct.env_factory.plans.new_plan()
261
262     return True
263
264 ### New Command
265 @callback(
266     Output("update_backend_coa", "data", allow_duplicate=True),
267     Input({"type": "button_add_new_command", "role": ALL}, "n_clicks"),
268     prevent_initial_call=True,
269 )
270 def new_command(nclicks):
271     logger.debug("BUTTON: new_command")
272     role = ctx.triggered_id['role']
273     current_plan = ct.env_factory.plans.current
274     current_plan.new_event(role)
275
276     locks["action_card"] = True
277
278     return True
279
280 ### Simulate Plan
281 @callback(
282     Output("update_backend_coa", "data", allow_duplicate=True),
283     Input("button_simulate_plan", "n_clicks"),
284     prevent_initial_call=True,
285 )
286 def simulate_plan(nclicks):
287     logger.debug("BUTTON: simulate_plan")
288     current_plan = ct.env_factory.plans.current
289     ct.env_factory.run_plan(current_plan)
290
291     return True
292
293
294
295 ### Generate Plan
296 @callback(
297     Output("update_backend_coa", "data", allow_duplicate=True),
298     Input("button_generate_plan", "n_clicks"),
299     prevent_initial_call=True,
300 )
301 def generate_plan(nclicks):
302     logger.debug("BUTTON: generate_plan")
303     ct.env_factory.generate_plan()
304     return True
305
306
307 #####
308 # Outputs
309 #####
310
311 @callback(
312     Output({"type": "onboard_model", "role": ALL}, "options", allow_duplicate=True),
313     Output({"type": "onboard_model", "role": ALL}, "value", allow_duplicate=True),

```

```

314     Output({"type": "timeline", "role": ALL}, "value", allow_duplicate=True),
315     Output({"type": "agent_interface", "role": ALL}, "value", allow_duplicate=True),
316     Output({"type": "actions_card", "role": ALL}, "children", allow_duplicate=True),
317     Output("table_plans", "children", allow_duplicate=True),
318     Output('button_generate_plan', 'disabled'),
319     Input("update_frontend_elements", "data"),
320     prevent_initial_call=True,
321 )
322
323 def display_choose_model(update):
324     logger.debug("OUTPUT: display_choose_model")
325
326     possible_models = ct.make_returns_from_dict(ct.env_factory.get_current_models())
327     timelines = ct.make_returns_from_dict(ct.env_factory.plans.current.get_dash_timelines())
328     interface = ct.make_returns_from_dict(ct.env_factory.plans.current.interfaces)
329     actions = actions_display(ct.env_factory)
330     plans = plan_menu(ct.env_factory)
331     generator = not ct.env_factory.generator_available()
332
333     selected_model = [ct.env_factory.plans.current.models[role] for role in ct.env_factory.roles]
334
335     logger.debug("\t %s %s", "Models", possible_models)
336     logger.debug("\t %s %s", "Selected Model", selected_model)
337     logger.debug("\t %s %s", "Timelines", timelines)
338     logger.debug("\t %s %s", "Interfaces", interface)
339
340     # return possible_models, Tb_children, timelines, interface
341     return possible_models, selected_model, timelines, interface, actions, plans, generator
342
343
344 ## Update COA Visualizations
345 @callback(
346     Output("plotly_visualization", "figure", allow_duplicate=True),
347     Input('update_frontend_elements', 'data'),
348     prevent_initial_call=True,
349 )
350
351 def visualization(update_frontend_elements):
352     logger.debug("OUTPUT: visualization")
353     if ct.env_factory.plans.current.visualizations is not None:
354         return ct.env_factory.plans.current.visualizations
355     else:
356         return no_update
357
358
359
360 ## Update Telemetry
361
362 @callback(
363     Output("stats-graphic-1", "figure", allow_duplicate=True),
364     Output("stats-graphic-2", "figure", allow_duplicate=True),
365     Output("stats-graphic-3", "figure", allow_duplicate=True),
366     Input('update_frontend_elements', 'data'),
367     prevent_initial_call=True,
368 )
369
370 def telemetry(update_frontend_elements):
371     logger.debug("OUTPUT: telemetry")
372     figures = []
373
374     for plan_id in ct.env_factory.plans.active:
375         logger.debug("\t %s %s", "active plan_id", plan_id)
376         plan = ct.env_factory.plans.get(plan_id)
377
378         if plan.telemetry is not None:
379             for i, t in enumerate(plan.telemetry[:3]):
380                 if len(figures) <= i:
381                     logger.debug("\t %s %s %s", "adding figure", len(figures), i)
382                     f = go.Figure()
383                     f.update_layout(
384                         margin={"l": 5, "b": 0, "t": 0, "r": 80},
385                         hovermode="closest",
386                         showlegend=False,
387                         xaxis_title=t.xlabel,
388                         yaxis_title=t.ylabel,
389                     )
390                     figures.append(f)
391
392                 t = t.as_df()
393                 opacity = .2
394                 if plan_id == ct.env_factory.plans.current.id:

```



```

395         logger.debug("\t %s %s %s", "active plan_id", plan_id, "this plan is the current
396             one.")
397         opacity = 1
398     for col in t.columns:
399         figures[i].add_trace(go.Scatter(x=t.index,y=t[col],
400             mode='lines',
401             name=f"{plan_id}_{col}",
402             opacity=opacity
403             ))
404
405     if len(figures) == 0:
406         return no_update
407
408     while len(figures)<3:
409         figures.append(go.Figure())
410
411     return figures
412
413 #####
414 # Update Backend
415 #####
416
417 @callback(
418     Output("update_frontend_elements", "data", allow_duplicate=True),
419     Input("update_backend_coa", "data"),
420     # prevent_initial_call=True,
421     prevent_initial_call='initial_duplicate'
422 )
423
424 def update_coa(coa_update):
425     logger.debug("BACKEND: update_coa")
426     return True
427
428 #####
429 # Setup
430 #####
431
432 locks = {
433     "action_card": False
434 }
435
436 if __name__ == "__main__":
437     logger.debug("app - ##### RELOADING DASH APP #####")
438     # external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
439     env_factory = env_creator(get_env_class)
440     ct.env_factory = env_factory
441
442     ## App Layout
443     app = Dash(__name__)
444     app.layout = app_layout(env_factory)
445     app.run(debug=True)

```

2.2 DASH/coach_integration.py

```
1  # Copyright (c) 2024 Mobius Logic, Inc.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #   http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import sys, inspect
16
17 from utilities.planning import COA, TimelineEvent
18 from coach import CommunicationSchedule
19 import plotly.express as px
20
21 from argparse import Namespace
22 from utilities.iotools import NumpyDecoder
23 import json
24 import os
25 import copy
26 import numpy as np
27 from itertools import product
28 import logging
29 logger = logging.getLogger(__name__)
30
31 from coach import (
32     SeededParameterGenerator,
33     BaseParameterGenerator,
34 )
35
36 import agents as agents_module
37 import directors as directors_module
38
39
40 def role_type(role):
41     return role.split("_")[0]
42
43 class COACHIntegration:
44     def __init__(self,
45                 env_creator,
46                 COACHEnvClass,
47                 parameters,
48                 agents_module=agents_module
49             ) -> None:
50         self.agents_module=agents_module
51         self.env_creator = env_creator
52         self.COACHEnvClass = COACHEnvClass
53         self.parameters = parameters
54         self.render_gif = False
55
56         self.library = dict()
57         self._id_to_interface = dict()
58         self._interface_to_id = dict()
59
60         self.setup()
61
62         self.plans = PlanLibrary(
63             self,
64             default_interfaces=self.default_interfaces,
65             default_models=self.default_models,
66         )
67
68         self.actions_container = {}
69
70     def get_current_models(self):
71         return {role: self.get_interface_models(interface) for role, interface in
72                 self.plans.current.interfaces.items()}
73
74     # def get_interfaces(self):
75     #     return list(self.interfaces.keys())
76
77     def set_current_interface(self, role, interface):
78         self.current_interface[role] = interface
```

```

78
79 def get_interfaces_by_role(self, role):
80     return list(self.interfaces_by_role[role].keys())
81
82 def get_current_interface(self, role):
83     return self.current_interface[role]
84
85 def get_interface_models(self, interface):
86     return list(self.model_params_by_interface[interface].keys())
87
88 def get_current_model(self):
89     return self.current_model
90
91 def set_current_model(self, role, model):
92     self.current_model[role] = model
93
94 def actions(self, role):
95     agt = self.coach_env.agents[role]
96     return agt.interface.action_dictionary
97
98 # Deal with Plan Library
99 def new_plan(self, name = None):
100     self.plans.new_plan(name)
101
102 def get(self, id):
103     return(self.library[id])
104
105 def keys(self):
106     return(self.library.keys())
107
108 def items(self):
109     return(self.library.items())
110
111 def values(self):
112     return(self.library.values())
113
114 def by_interface(self, interface):
115     return(self._interface_to_id[interface])
116
117 def get_by_interface(self, interface, id):
118     return(self._interface_to_id[interface][id])
119
120 def plot_COA(self, COA):
121
122     logger.debug("coach_integration.plot_COA - Generated COA Trajectory: %s", COA)
123     traj = self.coach_env.get_traj_from_coas({"player_0": COA})
124
125     logger.debug("coach_integration.plot_COA - Generated COA Trajectory: %s", len(traj))
126
127     return plot_trajectory_component(traj, {"player_0": COA}, self.coach_env.env)
128
129
130 def setup(self):
131     ## This should set up the game, get the information about it, and fix things like the
132     ## which roles can take which models.
133     ## COA_Env has agents, need to extract them from env.
134     # Create COA Env
135
136     self.COACH_params = COACH_params = self.parameters["COACH_params"]
137     self.env_params = env_params = self.parameters["env_params"]
138     self.actor_params = actor_params = self.parameters["actor_params"]
139
140     # Set up parameter generator
141     if COACH_params["stochastic"]:
142         self.parameter_generator = SeededParameterGenerator(23423, env_params)
143     else:
144         self.parameter_generator = BaseParameterGenerator(23423, env_params)
145
146     # Set up comm schedule
147     if "FIXED_STEPS_PER_COM" in COACH_params.keys():
148         schedule_param = COACH_params["FIXED_STEPS_PER_COM"]
149         self.comm_schedule = CommunicationSchedule.repeating(**schedule_param)
150     else:
151         logger.info("Communication Schedule Is Non-repeating")
152         self.comm_schedule = CommunicationSchedule(length=0)
153
154     self.ACTION_PADDING = COACH_params.get("ACTION_PADDING", 0)
155     self.MIN_NEXT_ACTION_TIME = COACH_params.get("MIN_NEXT_ACTION_TIME", 1)
156     self.MAX_NEXT_ACTION_TIME = COACH_params.get("MAX_NEXT_ACTION_TIME", np.inf)
157
158     self.coach_env = self.COACHEnvClass(

```

```

159         env_creator=self.env_creator,
160         parameter_generator=self.parameter_generator,
161         # agents=agent_dict,
162         fill_random=False,
163         comm_schedule=self.comm_schedule,
164         seed=COACH_params["seed"],
165     )
166
167     self.coach_env.augment(self.parameter_generator)
168     self.coach_env.reset()
169
170     self.roles = self.coach_env.env.agents
171     self.role_to_idx = {role:idx for idx, role in enumerate(self.roles)}
172     self.idx_to_role = {idx:role for idx, role in enumerate(self.roles)}
173
174     self.role_types = list(set([role_type(role) for role in self.roles]))
175     self.possible_models = self.coach_env.possible_models()
176
177
178     # Setup Actor Interfaces
179     self.interfaces_by_role = {role: dict() for role in self.roles}
180     self.roles_by_interface = dict()
181     self.interfaces = dict()
182     self.model_params_by_interface = dict()
183     self.model_params = dict()
184     self.model_classes = dict()
185     self.model_to_interface = dict()
186
187     for if_name, if_params in self.actor_params["interfaces"].items():
188         # Get the interface class
189         if_Class = self.agents_module.__dict__.get(if_params["interface_class"])
190         logger.debug("Current Class: %s", if_Class)
191         logger.debug("Current Params: %s", if_params)
192         # if_Class =
193             self.coach_env.__class__.agent_selection.Interfaces[if_params["interface_class"]]
194
195         # Get a reference interface, this is make sure that any model class instiated has the
196             correct interface
197         role = if_params.get("roles", self.roles)[0] # Get one applicable role
198         if_reference = if_Class(role, self.coach_env.env, **if_params.get("interface_parameters",
199             dict()))
200         logger.debug("Current Class: %s", if_reference)
201
202         # Set up references to interfaces
203         self.interfaces[if_name] = if_reference
204
205         # Get roles for which the interface is valid
206         for role in if_params.get("roles", self.roles):
207             self.interfaces_by_role[role][if_name] = if_reference
208             if if_name not in self.roles_by_interface.keys():
209                 self.roles_by_interface[if_name] = []
210             self.roles_by_interface[if_name].append(role)
211
212         # Setup reference to model creation information
213         self.model_params_by_interface[if_name] = if_params["models"]
214
215         self.model_classes[if_name] = dict()
216         for model_name, model_params in if_params["models"].items():
217             self.model_to_interface[model_name] = if_name
218             self.model_classes[if_name][model_name] =
219                 self.agents_module.__dict__.get(model_params["class_name"])
220             self.model_params[model_name] = model_params
221
222     # Setup Directors
223     directors = self.actor_params["directors"]
224     self.directors = dict()
225     self.directors_allow = dict()
226
227     for dr_name, dr_params in directors.items():
228
229         self.default_models = {role: params["classes"][0] for role, params in
230             dr_params['roles'].items()}
231         self.default_interfaces = {role: self.model_to_interface[model] for role, model in
232             self.default_models.items()}
233
234         tmp_params = copy.copy(self.parameters)
235
236         for role, params in dr_params["roles"].items():
237             tmp_params["COACH_params"]["Agents"][role] = self.model_params[params["classes"][0]]
238
239         dr_class = directors_module.__dict__.get(dr_params["class_name"])

```

```

234         self.directors[dr_name] = dr_class(
235             env_creator = self.env_creator,
236             COACHEnvClass = self.COACHEnvClass,
237             params = tmp_params,
238             model_path = dr_params["path"]
239         )
240
241         class_iter = product(*[classes["classes"] for classes in dr_params["roles"].values()])
242
243         for t in class_iter:
244             self.directors_allow[tuple(t)] = self.directors[dr_name]
245             self.model_to_interface
246
247     def generator_available(self):
248         return tuple(self.plans.current.models.values()) in self.directors_allow.keys()
249
250
251     def generate_plan(self):
252         current_models = self.plans.current.models
253         director = self.directors_allow[tuple(current_models.values())]
254         coas, traj = director.generate_coas(self.parameters)
255
256         tmp = self.plans.new_plan()
257         tmp.coas = coas
258         tmp.trajectory = traj
259
260         tmp.interfaces = copy.copy(self.plans.current.interfaces)
261         tmp.models = copy.copy(self.plans.current.models)
262         tmp.model_classes = copy.copy(self.plans.current.model_classes)
263         tmp.model_params = copy.copy(self.plans.current.model_params)
264
265
266     def run_plan(self, plan):
267         logger.debug("env_factory: run_plan")
268         logger.debug("\t plan info: %s %s %s", plan.name, plan.interfaces, plan.models)
269
270         if self.render_gif:
271             traj = self.coach_env.get_traj_from_plan(plan, render=True)
272             frames = np.array(traj.frames)
273             plan.visualizations = px.imshow(frames, animation_frame=0, binary_string=True,
274                                             labels=dict(animation_frame="slice"))
275         else:
276             traj = self.coach_env.get_traj_from_plan(plan, render=False)
277
278             plan.visualizations = self.coach_env.plot_trajectory_component(
279                 traj,
280                 plan=plan,
281                 env=self.coach_env.env,
282                 render_mode="plotly"
283             )
284
285             plan.trajectory = traj
286             plan.telemetry = self.coach_env.trajectory_telemetry(traj, plan, self.coach_env.env)
287
288 class Plan:
289     id = 0
290
291     @staticmethod
292     def fromCOAs(coas, name=None):
293         plan = Plan(coas.keys())
294         if name is not None:
295             plan.name = name
296
297         plan.coas = coas
298         return plan
299
300     def __init__(self,
301                 roles,
302                 name = None,
303                 interfaces_by_role=None,
304                 model_classes_library=None,
305                 model_params_library=None
306                 ):
307         self.id = copy.copy(Plan.id)
308         if name is None:
309             name = f"Plan_{Plan.id}"
310             Plan.id += 1
311
312         self.interfaces_by_role = interfaces_by_role
313         self.model_classes_library = model_classes_library

```

```

314         self.model_params_library = model_params_library
315
316         self.name = name
317         self.roles = list(roles)
318
319         self.coas = {role: COA() for role in self.roles}
320         self.interfaces = {role: None for role in self.roles}
321         self.models = {role: None for role in self.roles}
322         self.model_classes = {role: None for role in self.roles}
323         self.model_params = {role: dict() for role in self.roles}
324         self.trajectories = {role: None for role in self.roles}
325
326         self.visualizations = None
327         self.telemetry = None
328
329     def __getitem__(self, role):
330         return self.coas[role]
331
332     def items(self):
333         return self.coas.items()
334
335     def values(self):
336         return self.coas.values()
337
338     def keys(self):
339         return self.coas.keys()
340
341     def set_interface(self, role, interface):
342         if hasattr(interface, "__len__"):
343             if len(interface) == 0:
344                 interface = None
345             self.interfaces[role] = interface
346             self.models[role] = None
347             self.coas[role] = COA()
348
349     def get_interface(self, role):
350         interface = self.interfaces.get(role, None)
351         instance = self.interfaces_by_role[role][interface]
352
353         return interface, instance
354
355     def set_model(self, role, model):
356         self.models[role] = model
357         self.model_classes[role] = self.model_classes_library[self.interfaces[role]][model]
358         self.model_params[role] = self.model_params_library[self.interfaces[role]][model].get("params",
359                                                                                               dict())
360
361     def get_model(self, role):
362         return self.models[role]
363
364     def get_timelines(self):
365         return {role: self.coas[role].to_dict() for role in self.roles}
366
367     def get_dash_timelines(self):
368         timelines = self.get_timelines()
369         for k, v in timelines.items():
370             if len(v) == 0:
371                 timelines[k] = []
372             else:
373                 timelines[k] = list(v.keys())
374         return timelines
375
376     def new_event(self, role):
377         logger.debug("Adding new event to COA: %s", self.coas[role])
378         action_name, act_box = list(self.get_interface(role)[1].action_dictionary.items())[0]
379
380         ## [{"start":int, "action": str, "parameters":np.array}]
381         last_t = -1
382         if len(list(self.coas[role].timeline.keys()))>0:
383             last_t = list(self.coas[role].timeline.keys())[-1]
384
385         logger.debug("Action Name: %s", action_name)
386
387         tmp = TimelineEvent(
388             label=action_name,
389             parameters=copy.copy(act_box.default),
390             tags=role
391         )
392
393         self.coas[role].add_event(time=last_t + 1, event=tmp)

```

```

394
395         logger.debug("Added new event to COA: %s", self.coas[role])
396
397
398 class PlanLibrary:
399     def __init__(self,
400                 env_factory,
401                 plans = [],
402                 default_interfaces=None,
403                 default_models=None
404                 ):
405         self.roles = env_factory.roles
406         self._plans: list[Plan] = plans
407         self.current: Plan = None
408         self.active = set()
409         self.interfaces_by_role = env_factory.interfaces_by_role
410         self.model_classes_library = env_factory.model_classes
411         self.model_params_library = env_factory.model_params_by_interface
412
413         if not default_interfaces:
414             self.default_interfaces = {role: env_factory.get_interfaces_by_role(role)[0] for role in
415                                     self.roles}
416         else:
417             self.default_interfaces = default_interfaces
418
419         if not default_models:
420             self.default_models = {role:
421                                   env_factory.get_interface_models(self.default_interfaces[role])[0] for role in
422                                   self.roles}
423         else:
424             self.default_models = default_models
425
426         self.new_plan()
427         self.current = self.all()[0]
428         self.active.add(self.current.id)
429
430     def ids(self):
431         return list(self._id_to_plan.keys())
432
433     def get(self, id):
434         return self._id_to_plan[id]
435
436     def set_current_plan(self, id):
437         self.current = self._id_to_plan[id]
438
439     def new_plan(self, name=None):
440         tmp = Plan(
441             self.roles,
442             name=name,
443             interfaces_by_role = self.interfaces_by_role,
444             model_classes_library=self.model_classes_library,
445             model_params_library=self.model_params_library
446         )
447
448         for role, interface in self.default_interfaces.items():
449             tmp.set_interface(role, interface)
450             tmp.set_model(role, self.default_models[role])
451
452         self.add_plan(tmp)
453         return tmp
454
455     def add_plan(self, plan: Plan):
456         self._plans.append(plan)
457         self._id_to_plan = {plan.id:plan for plan in self._plans}
458
459     def all(self):
460         return self._plans
461
462
463 class COALibrary:
464     def __init__(self, roles, coas: dict[str, COA] = None):
465         self.roles = roles
466         self.role_types = list(set([role_type(role) for role in roles]))
467
468         self._library = {role_type:[] for role_type in self.role_types}
469
470         if coas is not None:
471             for role_tag, coa in coas.items:

```

```

472
473 def add_coa(self, role_tag, coa):
474     if role_tag in self.roles:
475         role_type = role_tag
476     elif role_type(role_tag) in self.roles:
477         role_type = role_type(role_tag)
478     else:
479         raise Exception(f"Role type {role_tag} cannot be found. Valid role types are
                        {list(self._library.keys())}.")
480
481     if coa not in self._library[role_type]:
482         self._library[role_type].append(coa)
483
484 def __setitem__(self, key, item):
485     self._library[key] = item
486
487 def __getitem__(self, key):
488     return self._library[key]
489
490 def values(self):
491     return self._library.values()
492
493 def items(self):
494     return self._library.items()
495
496 def keys(self):
497     return self._library.keys()

```


2.3 DASH/dash_utilities.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 class callback_tools:
16     env_factory = None
17
18     @staticmethod
19     def list_to_role(entry_list):
20         for i, entry in enumerate(entry_list):
21             if entry is not None:
22                 return i, callback_tools.env_factory.roles[i], entry
23
24     @staticmethod
25     def make_role_returns(role, r):
26         returns = [None]*len(callback_tools.env_factory.roles)
27         i = callback_tools.env_factory.roles.index(role)
28         returns[i] = r
29         return(returns)
30
31     @staticmethod
32     def get_role_from_callback(role, callback):
33         # print("Roles:", env_factory.role_to_idx)
34         return callback[callback_tools.env_factory.role_to_idx[role]]
35
36     @staticmethod
37     def make_returns_from_dict(r):
38         returns = [None]*len(callback_tools.env_factory.roles)
39         for i, role in enumerate(callback_tools.env_factory.roles):
40             if role in r.keys():
41                 returns[i] = r[role]
42
43         return callback_tools.nones_to_empty_list(returns)
44
45     @staticmethod
46     def make_returns(r):
47         return [r]*len(callback_tools.env_factory.roles)
48
49     @staticmethod
50     def nones_to_empty_list(r):
51         if type(r) is list:
52             return [callback_tools._ntol(v) for v in r]
53         if type(r) is dict:
54             return {k:callback_tools._ntol(v) for k, v in r.items()}
55
56     @staticmethod
57     def _ntol(r):
58         if r is None:
59             return []
60
61         return r
```

2.4 DASH/html_objects.py

```
1  # Copyright (c) 2024 Mobius Logic, Inc.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #   http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from dash import Dash, html, dcc, Input, Output, callback, State
16
17 def app_layout(env_factory):
18     ## The role interface select
19     RoleInterfaceSelect = []
20
21     for role in env_factory.roles:
22         RoleInterfaceSelect.append(html.Div(
23             [
24                 html.Div([
25                     role,
26                     dcc.Dropdown(
27                         env_factory.get_interfaces_by_role(role),
28                         placeholder=f"Interface",
29                         id={"type": "agent_interface", "role": role},
30                     ),
31                     dcc.Dropdown(
32                         id={"type": "onboard_model", "role": role},
33                         placeholder=f"Model",
34                     ),
35                     html.Button("Add New Command", id={"type": "button_add_new_command", "role":
36                                 role}),
37                 ],
38                 className="left_card",
39                 style={"float": "left", "margin": "auto"},
40             ),
41             html.Div(
42                 className="right_card",
43                 id={"type": "actions_card", "role": role},
44                 style={"float": "left", "margin": "auto"},
45             ),
46             className="row_agent_card",
47         ))
48
49     RoleInterfaceSelect = html.Div(RoleInterfaceSelect, className="action_cards")
50
51     ## The Role Timeline Selection
52     TimeLines = []
53     for role in env_factory.roles:
54         TimeLines += [
55             html.Tr([
56                 html.Td(role, style={"float": "left", "margin": "auto", "width": 100}),
57                 html.Td(
58                     dcc.RangeSlider(
59                         0,
60                         100,
61                         value=[],
62                         tooltip={"placement": "bottom", "always_visible": True},
63                         id={"type": "timeline", "role": role},
64                     ),
65                     style={"margin": "auto", "width": "100%"}
66                 ),
67             ])
68         ]
69
70     if len(TimeLines)==1:
71         TimeLines = TimeLines[0]
72
73     TimeLines = html.Table(TimeLines)
74
75     ## App Layout
76     app_layout = html.Div(
77         [
```

```

78         html.Div(
79             [
80                 html.Div(
81                     [
82                         dcc.Graph(
83                             id="plotly_visialization",
84                             style={"width": "600px", "height": "600px"},
85                         ),
86                     ],
87                     style={"width": "48%", "display": "inline-block"},
88                 ),
89                 html.Div(
90                     [
91                         dcc.Graph(id="stats-graphic-1", style={"height": "150px"}),
92                         dcc.Graph(id="stats-graphic-2", style={"height": "150px"}),
93                         dcc.Graph(id="stats-graphic-3", style={"height": "150px"}),
94                     ],
95                     style={"width": "48%", "float": "right", "display": "inline-block"},
96                 ),
97             ]
98         ),
99         TimeLines,
100         html.Button("New Plan", id="button_new_plan"),
101         html.Button("Generate Plan", id="button_generate_plan"),
102         html.Button("Simulate Plan", id="button_simulate_plan"),
103         ## These are True False values, they signle that the corresponding object should be updated.
104         dcc.Store(id="update_backend_coa"),
105         dcc.Store(id="update_frontend_elements"),
106         ### Agent Card:
107         html.Div(RoleInterfaceSelect),
108         html.Div([
109             "Plans",
110             html.Table(id="table_plans")
111         ], id="div_plans", style = {"width": "15%", "float": "right", "display": "inline-block",
112             "padding": "10px", "border": "1px solid black"})
113     ],
114 )
115 return app_layout
116
117 def action_table_row(time, event, interface, plan="", role=""):
118     CELL_WIDTH = "80px"
119
120     ## Action Label:
121     html_objects = []
122
123     ## Create the header describing the actions
124     action_head = [
125         html.Td("", style={"padding-left": "25px", "width": "200px"}),
126         html.Td("Time", style={"width": CELL_WIDTH, 'font-size': 10, 'font-style': 'italics'}),
127     ]
128
129     action_params = interface.action_dictionary[event["label"]]
130
131     for param in action_params.items():
132         # Give the name in the heading
133         action_head.append(html.Td(param.description, style={"width": CELL_WIDTH, 'font-size': 10,
134             'font-style': 'italics'}))
135
136     # Store header
137     html_objects.append(html.Tr(action_head))
138
139     TDs = [
140         html.Td(
141             [
142                 html.Div(
143                     className=f"rc-slider-handle rc-slider-handle-{time}",
144                     style={"margin-top": "0px", "margin-right": "5px"},
145                 ),
146                 html.Span(event["label"], style={"padding-left": "25px"}),
147             ],
148             style={"width": "200px"},
149         )
150     ]
151
152     TDs.append(
153         html.Td(
154             dcc.Input(
155                 id={
156                     "type": "actionstart",

```

```

157         "plan": plan,
158         "time": time,
159         "type": event['label'],
160         "role": role
161     },
162     type="number",
163     value=time,
164     placeholder=f"Step",
165     style={"width": CELL_WIDTH},
166     readOnly=True,
167     debounce = True,
168 )
169 )
170 )
171
172 # print("Event Parameters:", event["parameters"])
173
174 for i, param in enumerate(action_params.items()):
175     for j in range(param.shape[0]):
176         TDs.append(
177             html.Td(
178                 dcc.Input(
179                     id={
180                         "type": "actionparam",
181                         "plan": plan,
182                         "time": time,
183                         "type": event['label'],
184                         "index": i,
185                         "role": role
186                     },
187                     type="number",
188                     min=param.low[j],
189                     max=param.high[j],
190                     value=event["parameters"][i],
191                     placeholder=f"[{param.low[j]:.3}, {param.high[j]:.3}]", # the :.3 is the number
192                                     of significant figures
193                     style={"width": CELL_WIDTH},
194                     debounce = True,
195                 )
196             )
197
198 # print("Event Parameters:", TDs)
199 html_objects.append(html.Tr(TDs))
200
201 return html_objects
202
203
204 def actions_display(env_factory):
205     current_plan = env_factory.plans.current
206     html_objects = []
207     for role in env_factory.roles:
208         ## Render COA Actions
209         print("\t", f"Calling Model Card for {role}")
210
211         timelines = env_factory.plans.current.get_timelines()
212         interface_name, interface = current_plan.get_interface(role)
213
214         html_sub_objects = []
215
216         for i, (time, events) in enumerate(timelines[role].items()):
217             for j, event in enumerate(events.values()):
218                 html_sub_objects += action_table_row(time, event, interface, plan=current_plan.id,
219                                                         role=f"{role}")
220
221         html_objects.append(html.Div(html_sub_objects))
222
223 return html_objects
224
225
226 def plan_menu(env_factory):
227     html_objects = []
228     for plan in env_factory.plans.all():
229         if plan == env_factory.plans.current:
230             display_class = "current_plan"
231             style = {"background-color": "#ccc", "border": 0, "margin": 0, "padding": "10px 40px 10px 40px"}
232         else:
233             display_class = "archived_plan"
234             style = {"background-color": "#eee", "border": 0, "margin": 0, "padding": "10px 40px 10px 40px"}
235
236     print("\t", "plan_menu: plan", plan.name, display_class)

```

```

236
237     active = []
238     if plan.id in env_factory.plans.active:
239         active = [""]
240
241     html_objects.append( html.Tr(
242         [
243             html.Td(html.A(plan.name, id={"type": "plan", "id": plan.id}, n_clicks=0),
244                     className=display_class, style=style),
245             html.Td(dcc.Checklist([""], active, id={"type": "plan_view", "id": plan.id}))
246         ]
247     ))
248     return html.Tbody(html_objects)

```

3 DASH/Documentation

3.1 DASH/Documentation/interface.txt

```
1
2  ### Interface Selector
3  @callback(
4      Output("onboard_model", "options", allow_duplicate=True),
5      Output("model_table_body", "children", allow_duplicate=True),
6      Output("model_table_body", "value", allow_duplicate=True),
7      Input("agent_interface", "value"),
8      prevent_initial_call=True,
9  )
10 def choose_interface(value):
11     return [o._name_ for o in idx], Tb_children, []
12
13  ### Model Selector
14  @callback(
15      Output("model_table_head", "children", allow_duplicate=True),
16      Output("model_table_subhead", "children"),
17      Output("coa_update", "data", allow_duplicate=True),
18      Input("onboard_model", "value"),
19      prevent_initial_call=True,
20  )
21 def select_model(value):
22     return table_head, table_subhead, True
23
24
25  ### Add Action Button
26  @app.callback(
27      Output("my-range-slider", "value"),
28      Output("model_table_body", "children"),
29      Input("AddNewCommand", "n_clicks"),
30      State("my-range-slider", "value"),
31      State("model_table_body", "children"),
32      State("agent_interface", "value"),
33  )
34 def add_COA_command(val, slider_values, children, interface):
35     return slider_values, children
36
37
38  ### Slider Action
39  @callback(
40      Output({"type": "actionstart", "index": ALL}, "value"),
41      Output("coa_update", "data", allow_duplicate=True),
42      Input("my-range-slider", "drag_value"),
43      State("agent_interface", "value"),
44      prevent_initial_call=True,
45  )
46 def update_output(drag_value, interface):
47     return drag_value, True
48
49
50  ### Parameter Update
51  @callback(
52      Output("coa_update", "data", allow_duplicate=True),
53      Input({"type": "actionparam", "index": ALL}, "value"),
54      State("agent_interface", "value"),
55      prevent_initial_call=True,
56  )
57 def update_coa_parameters(value, interface):
58     return interface
59
60
61  ### Process COA Update
62  @callback(
63      Output("indicator-graphic", "figure"),
64      Output("stats-graphic-vel", "figure"),
65      Output("stats-graphic-fuel", "figure"),
66      Output("stats-graphic-acc", "figure"),
67      Input("coa_update", "data"),
68      State("agent_interface", "value"),
69  )
70 def process_coa_update(value, interface):
71     return fig, fig1, fig2, fig3
72
73
74  ### Generate COA
75  @callback(
```

```

76     Output("memory", "data"),
77     Input("GenerateCOA", "n_clicks"),
78     State("agent_interface", "value"),
79     prevent_initial_call=True,
80 )
81 def generate_coa(value, interface):
82     return "generated"
83
84
85 ### Parameter Update
86 @callback(
87     Output("model_table_body", "children", allow_duplicate=True),
88     Output("my-range-slider", "value", allow_duplicate=True),
89     Output("coa_update", "data", allow_duplicate=True),
90     Input("memory", "data"),
91     State("model_table_body", "children"),
92     State("my-range-slider", "value"),
93     State("agent_interface", "value"),
94     prevent_initial_call=True,
95 )
96 def load_coa(coa_name, table_body, slider, interface):
97     return table_body, slider_values, True

```

3.2 DASH/Documentation/network_view.py

```
1  # %%
2  import networkx as nx
3  import matplotlib.pyplot as plt
4
5  s = ""
6  with open("../app.py", "r") as f:
7      in_callback = False
8      callbacks = []
9      s = ""
10
11     for line in f:
12         if in_callback:
13             s += line
14
15             if line.startswith("def"):
16                 in_callback = False
17                 callbacks.append((line, s))
18
19
20     if line.startswith("@callback"):
21         s = ""
22         in_callback = True
23
24
25 # %%
26 info = dict()
27
28
29 for d, c in callbacks:
30     Outputs = []
31     Input = []
32     State = []
33
34     c = c.strip()
35     cs = c.split(",")
36     i = 0
37     while i < len(cs):
38         st = cs[i].strip()
39
40         if st.startswith("Output"):
41             target = st.split("(")[1].replace("'", "").replace('"', "")
42             atr = cs[i+1].replace("'", "").replace('"', "").replace(")", "")
43             Outputs.append([target, atr])
44             i+=1
45
46         if st.startswith("Input"):
47             target = st.split("(")[1].replace("'", "").replace('"', "")
48             atr = cs[i+1].replace("'", "").replace('"', "").replace(")", "")
49             Input.append([target, atr])
50             i+=1
51
52         if st.startswith("State"):
53             target = st.split("(")[1].replace("'", "").replace('"', "")
54             atr = cs[i+1].replace("'", "").replace('"', "").replace(")", "")
55             State.append([target, atr])
56             i+=1
57
58     i+=1
59
60     name = d.split()[1].split("(")[0]
61     info[name] = {
62         "Input": Input,
63         "Output": Outputs,
64         "State": State
65     }
66
67 info
68 # %%
69 edges = []
70 functions = []
71 inputs = []
72 outputs = []
73 states = []
74 for name, s in info.items():
75     functions.append(name)
76
77     for i in s["Input"]:
```



```

79         inputs.append(i[0])
80         edges.append( (i[0], name) )
81
82     for i in s["Output"]:
83         outputs.append(i[0])
84         edges.append( (name, i[0]) )
85
86     for i in s["State"]:
87         states.append(i[0])
88         edges.append( (i[0], name) )
89
90     edges
91     # %%
92     G = nx.Graph().to_directed()
93
94     # %%
95     color_map = []
96     for n in G:
97         n = str(n)
98         if n in functions:
99             color_map.append("blue")
100         elif n in inputs:
101             color_map.append("green")
102         elif n in outputs:
103             color_map.append("red")
104         elif n in states:
105             color_map.append("orange")
106         else:
107             color_map.append("black")
108
109     color_map
110
111     # %%
112     for e in edges:
113         G.add_edge(e[0], e[1])
114     # G = nx.from_edgelist(edges).to_directed()
115
116     f = plt.figure(figsize=[12,12])
117     ax = f.gca()
118     pos = nx.nx_agraph.graphviz_layout(G, prog="dot")
119     nx.draw(
120         G,
121         pos=pos,
122         ax=ax,
123         with_labels=False,
124         node_color=color_map
125     )
126     text = nx.draw_networkx_labels(G, pos)
127     for _, t in text.items():
128         t.set_rotation(20)
129
130     print(text)
131     # %%

```

4 DASH/assets

4.1 DASH/assets/typography.css

```
1 body {
2     font-family: sans-serif;
3 }
4
5 h1, h2, h3, h4, h5, h6 {
6     color: black
7 }
8
9 .resume {
10     display: none;
11 }
12
13 .button4 {
14     background-color: white; /* Green */
15     border: none;
16     color: black;
17     padding: 15px 32px;
18     text-align: left;
19     text-decoration: none;
20     display: inline-block;
21     font-size: 16px;
22     width: 100%;
23 }
24
25 .button4:hover {
26     background-color: #DDDDDD; /* Green */
27     color: white;
28 }
29
30
31 .float-container {
32     border: 3px solid #fff;
33     padding: 20px;
34     width: 600px;
35     margin: 0;
36     height: 300px;
37 }
38
39 .float-child {
40     width: 50%;
41     height: 100%;
42     float: left;
43     padding: 20px;
44     border: 2px solid red;
45 }
46
47 .agent_card {
48     float: left;
49     padding: 10px;
50     height: 150px;
51     width: 100%;
52     margin: auto;
53 }
54
55 .action_cards {
56     float: left;
57     padding: 10px;
58     width: 80%;
59     margin: auto;
60 }
61
62 .left_card {
63     float: left;
64     width: 25%;
65     height: 100%;
66     background-color: #aaa;
67     margin: auto;
68 }
69
70 .right_card {
71     float: left;
72     width: 70%;
73     height: 100%;
74     background-color: #bbb;
75     margin: auto;
```

```

76     }
77
78     .coa_card {
79         float: left;
80         width: 15%;
81         height: 100%;
82         background-color: #bbb;
83         margin:auto;
84     }
85
86     .current_plan tr {
87         background-color: #ccc;
88     }
89
90     .archived_plan tr {
91         background-color: #eee;
92     }
93
94     .archived_plan tr:hover {
95         background-color: #ddd;
96     }
97
98     /* Clear floats after the columns */
99     .agent_card_container:after {
100         content: "";
101         display: table;
102         clear: both;
103     }
104
105
106     /* Colors for sliders */
107
108     .rc-slider-handle-1 {
109         border-color: #01befe;
110     }
111
112     .rc-slider-handle-1.rc-slider-handle-click-focused:focus {
113         border-color: #01befe;
114     }
115
116     .rc-slider-handle-1:hover {
117         border-color: #01befe;
118     }
119
120     /* ===== */
121
122     .rc-slider-handle-2 {
123         border-color: #ffdd00;
124     }
125
126     .rc-slider-handle-2.rc-slider-handle-click-focused:focus {
127         border-color: #ffdd00;
128     }
129
130     .rc-slider-handle-2:hover {
131         border-color: #ffdd00;
132     }
133
134     /* ===== */
135
136     .rc-slider-handle-3 {
137         border-color: #ff7d00;
138     }
139
140     .rc-slider-handle-3.rc-slider-handle-click-focused:focus {
141         border-color: #ff7d00;
142     }
143
144     .rc-slider-handle-3:hover {
145         border-color: #ff7d00;
146     }
147
148     /* ===== */
149
150     .rc-slider-handle-4 {
151         border-color: #ff006d;
152     }
153
154     .rc-slider-handle-4.rc-slider-handle-click-focused:focus {
155         border-color: #ff006d;
156     }

```

```

157
158 .rc-slider-handle-4:hover {
159     border-color: #ff006d;
160 }
161
162 /* ===== */
163
164 .rc-slider-handle-5 {
165     border-color: #adff02;
166 }
167
168 .rc-slider-handle-5.rc-slider-handle-click-focused:focus {
169     border-color: #adff02;
170 }
171
172 .rc-slider-handle-5:hover {
173     border-color: #adff02;
174 }
175
176 /* ===== */
177
178 .rc-slider-handle-6 {
179     border-color: #8f00ff;
180 }
181
182 .rc-slider-handle-6.rc-slider-handle-click-focused:focus {
183     border-color: #8f00ff;
184 }
185
186 .rc-slider-handle-6:hover {
187     border-color: #8f00ff;
188 }
189
190 /* ===== */

```

5 examples/MAInspection

5.1 examples/MAInspection/Readme.md

```
1 # Multiagent Inspection
2
3 This example shows how to solve a simple satellite inspection problem in two steps:
4 * First we train an agent not to solve the problem, but to use the environment to train an agent to go
   from one waypoint to another.
5 * After this "waypointer"
6 agent is trained the director is then trained to select waypoints that easily solve the problem.
7
8 This example also demonstrates how to create custom agents, a custom coach environment, custom
   trajectory classes and custom trajectory visualizations.
9
10 
```

5.2 examples/MAInspection/app.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #   http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import sys
16 import glob
17 import yaml
18 sys.path.insert(0, "../..")
19
20 from DASH.app import *
21 from utilities.PZWrapper import PettingZooEnv
22
23 from examples.MAInspection.Environments import env as Multinspect
24 from examples.MAInspection.Environments.MAIcoach import MAI_COACH
25 import examples.MAInspection.Environments.MAIagents as MAIagents
26
27 # Logging
28 import logging
29 logging.basicConfig(level=logging.DEBUG, format="%(levelname)s: %(name)s: %(message)s")
30 logger = logging.getLogger(__name__)
31
32 pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
33                   name.startswith("pymunk")]
34 numba_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
35                  name.startswith("numba")]
36 for log_handler in pymunk_loggers + numba_loggers:
37     log_handler.setLevel(logging.INFO)
38
39 def get_env():
40     return PettingZooEnv(PZGame=Multinspect)
41
42 if __name__ == "__main__":
43     print("app - ##### RELOADING DASH APP #####")
44
45     with open(os.path.join("Experiment", "train_director.yaml"), "r") as f:
46         try:
47             params = yaml.safe_load(f)
48         except yaml.YAMLError as exc:
49             print(exc)
50
51     with open(os.path.join("Experiment", "dash_app.yaml"), "r") as f:
52         try:
53             params["actor_params"] = yaml.safe_load(f)
54         except yaml.YAMLError as exc:
55             print(exc)
56
57     # external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
58     env_factory = COACHIntegration(
59         env_creator=get_env,
60         COACHEnvClass=MAI_COACH,
61         parameters=params,
62         agents_module=MAIagents,
63     )
64
65     ct.env_factory = env_factory
66
67     locks = {
68         "action_card": False
69     }
70
71     ## App Layout
72     proxy_url = re.sub("{{port}}", "8050", os.environ["VSCODE_PROXY_URI"])
73     proxy_url = re.sub(os.environ["ACEHUB_BASEURL"], "", proxy_url)
74     app = Dash(serve_locally=True, requests_pathname_prefix=proxy_url)
75
76     app.layout = app_layout(env_factory)
```

```
77  
78     app.run(debug=True)
```

5.3 examples/MAInspection/train_agents.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #   http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from __future__ import annotations
16 import sys
17 sys.path.insert(0, "../..")
18
19 import examples.MAInspection.Environments.env as TrainingEnv
20 import supersuit as ss
21 from stable_baselines3 import PPO
22 from stable_baselines3.ppo import MlpPolicy
23 from stable_baselines3.common.callbacks import EvalCallback
24 from stable_baselines3.common.vec_env import VecVideoRecorder
25 from PIL import Image
26 import os
27 import time
28 import glob
29 from tqdm import tqdm
30 import numpy as np
31 import yaml
32 import copy
33
34 import torch
35 CORES = 7
36 torch.set_num_threads(CORES)
37 torch.set_num_interop_threads(CORES)
38
39 # Code adapted from https://pettingzoo.farama.org/tutorials/sb3/waterworld/
40
41 ## We're going to train two policies, one that assumes a densely poisonous env
42 ## and one that requires sparse exploration.
43
44 def train_butterfly_supersuit(
45     env_fn,
46     steps: int = 10_000,
47     seed: int | None = 0,
48     num_vec_envs=1,
49     num_cpus=CORES,
50     learning_rate=1e-3,
51     batch_size=256,
52     model_dir="",
53     name="",
54     **env_kwargs
55 ):
56     model_path = os.path.join(model_dir, name)
57     os.makedirs(model_path, exist_ok=True)
58
59     # Train a single model to play as each agent in a cooperative Parallel environment
60     env = env_fn(**env_kwargs)
61     env.reset(seed=seed)
62
63     print(f"Starting training on {str(env.metadata['name'])}.")
64
65     env = ss.pettingzoo_env_to_vec_env_v1(env)
66     env = ss.concat_vec_envs_v1(env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
67                                base_class="stable_baselines3")
68
69     # Note: Waterworld's observation space is discrete (242,) so we use an MLP policy rather than CNN
70     model = PPO(
71         MlpPolicy,
72         env,
73         verbose=3,
74         learning_rate=learning_rate,
75         batch_size=batch_size,
76         tensorboard_log=os.path.join("../tensorboard_log/", name)
77     )
```



```

78     eval_env = env_fn(**env_kwargs)
79     eval_env.reset(seed=seed)
80     eval_env = ss.pettingzoo_env_to_vec_env_v1(eval_env)
81     eval_env = ss.concat_vec_envs_v1(eval_env, num_vec_envs=1, num_cpus=num_cpus,
      base_class="stable_baselines3")
82
83     eval_callback = EvalCallback(eval_env, verbose=1, eval_freq=1000)
84
85     weight_name = f"{env.unwrapped.metadata.get('name')}_{'time.strftime('%Y%m%d-%H%M%S')}"
86     weight_path = os.path.join(model_path, weight_name)
87     os.makedirs(weight_path, exist_ok=True)
88
89     model_args = {
90         "name": name,
91         "load_class": "SB_PP0WaypointerActor",
92         "env_params": env_kwargs
93     }
94
95     with open(os.path.join(weight_path, "params.yaml"), "w") as f:
96         yaml.dump(model_args, f, default_flow_style=False)
97
98     model.learn(total_timesteps=steps, callback=eval_callback)
99     model.save(os.path.join(weight_path, "model"))
100
101     print("Model has been saved.")
102     print(f"Finished training on {str(env.unwrapped.metadata['name'])}.")
103
104     env.close()
105
106     return os.path.join(weight_path, "model")
107
108
109 def eval(
110     env_fn,
111     model_path,
112     num_games: int = 5,
113     render_mode: str | None = "rgb_array",
114     render = True,
115     **env_kwargs
116 ):
117     # Evaluate a trained agent vs a random agent
118     env = env_fn(render_mode=render_mode, **env_kwargs)
119
120     print(
121         f"\nStarting evaluation on {str(env.metadata['name'])} (num_games={num_games},
           render_mode={render_mode})"
122     )
123
124     print(model_path)
125
126     model = PP0.load(model_path)
127
128     total_rewards = []
129
130     # Note: We train using the Parallel API but evaluate using the AEC API
131     # SB3 models are designed for single-agent settings, we get around this by using the same model for
     every agent
132     for i in tqdm(range(num_games)):
133         obs, infos = env.reset(seed=i)
134         cumm_rewards = {agent: 0 for agent in env.possible_agents}
135         running = True
136         frames = []
137
138         while running:
139             acts = dict()
140             for role in env.agents:
141                 acts[role] = model.predict(obs[role], deterministic=True)[0]
142
143             obs, rewards, terminations, truncations, infos = env.step(acts)
144
145             for role in env.agents:
146                 cumm_rewards[role] += rewards[role]
147
148             if all([a or b for a, b in zip(terminations.values(), truncations.values())]):
149                 running = False
150                 total_rewards += list(cumm_rewards.values())
151
152             if render:
153                 frames.append(env.render())
154
155     if render:

```

```

156         imgs = [Image.fromarray(frame) for frame in frames]
157         # duration is the number of milliseconds between frames; this is 40 frames per second
158         model_dir = model_path.split(".")[0]
159         imgs[0].save(model_dir + f"_eval_run_{i}.gif", save_all=True, append_images=imgs[1:],
160                     duration=500, loop=0)
161
162     env.close()
163
164     avg_reward = np.mean(total_rewards)
165     std_reward = np.std(total_rewards)
166
167     with open(model_path + ".txt", "a") as f:
168         f.writelines(f"\n\n{env_kwargs}\n")
169         f.writelines(f"\t Avg reward: {avg_reward}, std: {std_reward}\n")
170         f.writelines(f"\t Rewards: {total_rewards}\n")
171
172     print(f"\t Avg reward: {avg_reward}, std: {std_reward}")
173     return avg_reward
174
175 if __name__ == "__main__":
176
177     with open("Experiment/train_agents.yaml", "r") as stream:
178         try:
179             experiments = yaml.safe_load(stream)
180         except yaml.YAMLError as exc:
181             print(exc)
182
183     STEPS = 2000
184
185     # Train Waypoint Agent
186     model_paths = dict()
187     for name, experiment in experiments.items():
188         model_paths[name] = train_butterfly_supersuit(
189             env_fn=TrainingEnv.parallel_env,
190             steps=STEPS,
191             model_dir=name,
192             **experiment["env_params"]
193         )
194
195     for name, model_path in model_paths.items():#, model_path_explore]:
196         eval(env_fn=TrainingEnv.parallel_env,
197             model_path=model_path,
198             **experiments[name]["env_params"]
199         )

```

5.4 examples/MAInspection/train_director.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from __future__ import annotations
16 import os
17 import time
18 import sys
19 import glob
20 from tqdm import tqdm
21 import numpy as np
22 from PIL import Image
23 import yaml
24
25 sys.path.insert(0, "../..")
26
27 from utilities.PZWrapper import PettingZooEnv
28 from examples.MAInspection.Environments.MAIcoach import MAI_COACH
29 from examples.MAInspection.Environments import env as Multinspect
30 from env import COACH_PettingZoo
31
32 from stable_baselines3 import PPO
33 from stable_baselines3.ppo import MlpPolicy
34 from stable_baselines3.common.callbacks import EvalCallback
35 import supersuit as ss
36
37 import torch
38 CORES = 6
39 torch.set_num_threads(CORES)
40 torch.set_num_interop_threads(CORES)
41
42 import logging
43 logging.basicConfig()
44 logging.getLogger().setLevel(logging.INFO)
45 logger = logging.getLogger(__name__)
46
47 pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
48                   name.startswith("pymunk")]
49 for log_handler in pymunk_loggers:
50     log_handler.setLevel(logging.INFO)
51
52 # SB code adapted from https://pettingzoo.farama.org/tutorials/sb3/waterworld/
53
54 ## We're going to train two policies, one that assumes a densely poisonous env
55 ## and one that requires sparse exploration.
56 # %%
57 def train(
58     env_fn,
59     steps: int = 10_000,
60     seed: int | None = 0,
61     num_vec_envs=1,
62     num_cpus=1,
63     learning_rate=1e-3,
64     batch_size=256,
65     model_dir=""
66 ):
67     os.makedirs(model_dir, exist_ok=True)
68
69     # Train a single model to play as each agent in a cooperative Parallel environment
70     env = env_fn()
71     env.reset(seed=seed)
72     print(env.possible_agents)
73
74     logger.info(f"Starting training on {str(env.metadata['name'])}.")
75
76     env = ss.pettingzoo_env_to_vec_env_v1(env)
77     env = ss.concat_vec_envs_v1(env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
78                                base_class="stable_baselines3")
```

```

77
78     model = PP0(
79         MlpPolicy,
80         env,
81         verbose=3,
82         learning_rate=learning_rate,
83         batch_size=batch_size,
84         tensorboard_log=os.path.join("./tensorboard_log/", model_dir.split("/")[-1])
85     )
86
87     eval_env = env_fn()
88     eval_env = ss.pettingzoo_env_to_vec_env_v1(eval_env)
89     eval_env = ss.concat_vec_envs_v1(eval_env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
90         base_class="stable_baselines3")
91     eval_callback = EvalCallback(eval_env, verbose=1, eval_freq=500)
92
93     model.learn(total_timesteps=steps, callback=eval_callback)
94
95     model_name = f"{env.unwrapped.metadata.get('name')}__{time.strftime('%Y%m%d-%H%M%S')}"
96     model_path = os.path.join(model_dir, model_name)
97     model.save(model_path)
98
99     logger.info("Model has been saved.")
100     logger.info(f"Finished training on {str(env.unwrapped.metadata['name'])}.")
101
102     env.close()
103
104     return model_path + ".zip"
105
106 def eval(
107     env_fn,
108     model_path,
109     num_games: int = 10,
110     render = False,
111     **env_kwargs
112 ):
113     # Evaluate a trained agent vs a random agent
114     env = env_fn(**env_kwargs)
115
116     logger.info(
117         f"\nStarting evaluation on {str(env.metadata['name'])} (num_games={num_games}, render={render})"
118     )
119
120     model = PP0.load(model_path)
121
122     cum_rewards = []
123
124     # Note: We train using the Parallel API but evaluate using the AEC API
125     # SB3 models are designed for single-agent settings, we get around this by using the same model for every agent
126     for i in tqdm(range(num_games)):
127         if render:
128             env.coa_env.start_rendering()
129
130         obs, info = env.reset(seed=i)
131
132         rewards = {agent: 0 for agent in env.possible_agents}
133         running = True
134         j = 0
135         while running:
136             j += 1
137             act = {"director": model.predict(obs["director"], deterministic=False)[0]}
138
139             logger.debug("Obs From Director: %s", obs["director"]) # DEBUG
140             logger.debug("Action From Director: %s", act) # DEBUG
141
142             obs, reward, term, trunc, info = env.step(act)
143             for a in env.agents:
144                 rewards[a] += reward[a]
145
146             if all([a or b for a,b in zip(term.values(), trunc.values())]):
147                 for a in env.agents:
148                     cum_rewards.append(rewards[a])
149
150                 running = False
151
152         if render:
153             frames = env.coa_env.state.trajecory.frames
154             imgs = [Image.fromarray(frame) for frame in frames]
155             # duration is the number of milliseconds between frames; this is 40 frames per second
156             model_dir = model_path.split(".")[0]

```

```

156         imgs[0].save(model_dir + f"_eval_run_{i}.gif", save_all=True, append_images=imgs[1:],
157                       duration=500, loop=0)
158     env.close()
159
160     avg_reward = np.mean(cum_rewards)
161     std_reward = np.std(cum_rewards)
162
163     with open(model_path + ".txt", "a") as f:
164         f.writelines(f"\n\n{env_kwargs}\n")
165         f.writelines(f"\t Avg reward: {avg_reward}, std: {std_reward}\n")
166         f.writelines(f"\t Rewards: {cum_rewards}\n")
167
168     logger.info(f"\t Avg reward: {avg_reward}, std: {std_reward}")
169     return avg_reward
170
171
172 def get_env():
173     return PettingZooEnv(PZGame=Multinspect)
174
175 if __name__ == "__main__":
176     WAYPOINTER_DIR = "waypointer"
177     WAYPOINTER_PATH = min(glob.iglob('waypointer/*'), key=os.path.getctime)
178
179     WAYPOINTER_PATH = min(glob.iglob(os.path.join(WAYPOINTER_PATH, '*.zip')), key=os.path.getctime)
180
181     with open(os.path.join("Experiment", "train_director.yaml"), "r") as stream:
182         try:
183             params = yaml.safe_load(stream)
184         except yaml.YAMLError as exc:
185             print(exc)
186
187     for agent, param in params["COACH_params"]["Agents"].items():
188         param["params"]["policy_path"] = WAYPOINTER_PATH
189
190     env = COACH_PettingZoo(env_creator=get_env, COACHEnvClass=MAI_COACH)
191
192     def get_env_pz():
193         env = COACH_PettingZoo(env_creator=get_env, COACHEnvClass=MAI_COACH)
194         env.augment(params)
195         env.reset()
196         return env
197
198     model_path = train(
199         get_env_pz,
200         steps=500,
201         seed=0,
202         model_dir="director"
203     )
204
205     model_path = min(glob.iglob(os.path.join('director', '*.zip')), key=os.path.getctime)
206     rew = eval(
207         get_env_pz,
208         model_path=model_path,
209         num_games=1,
210         render=True
211     )
212
213     with open(os.path.join("Experiment", "dash_app_template.yaml"), 'r') as f:
214         with open(os.path.join("Experiment", "dash_app.yaml"), 'w') as g:
215             for line in f.readlines():
216                 line = line.replace("<WAYPOINTER_PATH>", WAYPOINTER_PATH)
217                 line = line.replace("<DIRECTOR_PATH>", model_path)
218                 g.write(line)
219
220 # %%

```

6 examples/MAInspection/Environments

6.1 examples/MAInspection/Environments/MAIagents.py

```
1  # Copyright (c) 2024 Mobius Logic, Inc.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #   http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import numpy as np
16 import copy
17 import logging
18 from gymnasium.spaces import Box
19 logger = logging.getLogger(__name__)
20 from numpy.random import PCG64DXSM, Generator
21 from agents import *
22
23 class WaypointInterface(TrivialInterface):
24     def __init__(self, role, env):
25         super().__init__(role, env)
26
27         desc = ["rel frame x", "rel frame y", "rel frame z"]
28
29         policies = ActionBox(
30             low=-np.ones(3)*env._MAX_OUTER_PERIMETER.value,
31             high=np.ones(3)*env._MAX_OUTER_PERIMETER.value,
32             shape=(3,),
33             description=desc
34         )
35
36         self.action_dictionary = {"Waypoint": policies}
37
38     def equals(self, other):
39         if type(other) is type(self):
40             return self.action_dictionary["Waypoint"].equals(other.action_dictionary["Waypoint"])
41         return False
42
43
44 class SB_PPOWaypointActor(BasicActor):
45     InterfaceType = WaypointInterface
46     def __init__(self, role, policy_path: str, interface:WaypointInterface=None):
47         super().__init__(role)
48
49         if policy_path[-3:] == ".zip":
50             policy_path = policy_path.split(".")[0]
51         self.policy = PPO.load(policy_path)
52         self.next_waypoint = None
53
54     def __str__(self):
55         return f"SB_PPOWaypointActor: {self.role}"
56
57     def get_action(self, obs, t, mean_mode=False):
58         bad_command = False
59         acting = False
60         coa_done = False
61         action = copy.copy(self.none_action)
62
63         if t in self.coa.timeline.keys():
64             ## Start new waypoint
65             e = self.coa.get(time = t, label = "Waypoint")
66             # self.time_remaining = e.parameters
67             self.next_waypoint = e.parameters
68             logger.debug("Agent: start new waypoint %s", e) # DEBUG
69
70
71         if self.next_waypoint is not None:
72             i = self.env.possible_agents.index(self.role)
73
74             pos = self.env.orb[self.role].r.to(self.env._OU_DIS).value
75             pos_c = self.env.orb["chief"].r.to(self.env._OU_DIS).value
```

```

76
77         if self.env.OBSERVATION_FRAME == "Hills":
78             frame = self.env.hills_frame(self.env.orb[self.role])
79         else:
80             frame = self.env.ori[i]
81
82         rel_waypt = frame.T @ (self.next_waypoint + pos_c - pos).reshape(-1,1)
83
84         agent_obs = np.concatenate([obs[:6*self.env.num_deputies],rel_waypt.reshape(-1)])
85         if np.linalg.norm(rel_waypt) < self.env._WAYPOINT_ARRIVAL_PROX.value:
86             acting = False
87             coa_done = True
88             self.next_waypoint = None
89         else:
90             acting = True
91             action, _ = self.policy.predict(agent_obs, deterministic=True)
92
93         return action, {
94             "acting": acting,
95             "coa_done": coa_done,
96             "bad_command": bad_command,
97         }
98
99     def reset(self, env):
100         super().reset(env)
101         self.time_remaining = -1
102         self.current_policy = None
103
104         self.interface = SB_PP0WaypointActor.InterfaceType(
105             self.role,
106             env
107         )
108
109         self.env = env
110
111         if self.reference_interface is not None:
112             if not self.interface.equals(self.reference_interface):
113                 raise Exception("Actor interface does not match reference interface.")
114
115
116     classes = [
117         cls_obj
118         for cls_name, cls_obj in inspect.getmembers(sys.modules[__name__])
119         if inspect.isclass(cls_obj) and cls_obj.__module__ == __name__
120     ]
121
122     Interfaces = {}
123     for c in classes:
124         # BasicActor
125         if issubclass(c, TrivialInterface):
126             Interfaces[c] = []
127
128     Agents = {}
129     for c in classes:
130         # BasicActor
131         if issubclass(c, BasicActor):
132             Interfaces[c.InterfaceType].append(c)
133             Agents[c.__name__] = c

```

6.2 examples/MAInspection/Environments/MAIcoach.py

```
1  # Copyright (c) 2024 Mobius Logic, Inc.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #     http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import sys
16 sys.path.insert(0, "../..")
17
18 import numpy as np
19 import copy
20 import logging
21 from gymnasium.spaces import Box
22
23 logger = logging.getLogger(__name__)
24 from collections import OrderedDict
25 import matplotlib.pyplot as plt
26 from numpy.random import PCG64DXSM, Generator
27 import matplotlib as mpl
28 from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
29 from mpl_toolkits.mplot3d.art3d import Line3DCollection
30 import plotly.express as px
31 import plotly.graph_objects as go
32 import pandas as pd
33 import examples.MAInspection.Environments.MAIagents as agents
34 import io
35 from PIL import Image
36
37 # %matplotlib inline
38 from coach import (
39     COACHEnvironment,
40     COA,
41     State,
42     Trajectory,
43     BaseParameterGenerator,
44     SeededParameterGenerator,
45     CommunicationSchedule,
46     Timeline
47 )
48
49 # %%
50 def plot_orbits(orbits):
51     trace = go.Scatter3d(
52         x=orbits[:, 0],
53         y=orbits[:, 1],
54         z=orbits[:, 2],
55         mode="markers",
56         marker=dict(color=orbits[:, 3:], size=5),
57     )
58
59     return trace
60
61     # ax.scatter(orbits[:,0], orbits[:,1], orbits[:,2], c = orbits[:,3:], alpha = alpha)
62
63
64 def plot_points(p_array, c):
65     # c = np.zeros([seen.shape[0], 3])
66     c = mpl.colors.to_rgba_array(c)
67     tmp = np.zeros([p_array.shape[0], 7])
68     tmp[:, :3] = p_array
69     tmp[:, 3:] = c
70
71     t_seen = go.Scatter3d(
72         x=tmp[:, 0],
73         y=tmp[:, 1],
74         z=tmp[:, 2],
75         mode="markers",
76         marker=dict(color=tmp[:, 3:], size=5),
77     )
78
```



```

79         return t_seen
80
81
82     def ms(x, y, z, radius, resolution=20):
83         """Return the coordinates for plotting a sphere centered at (x,y,z)"""
84         u, v = np.mgrid[0 : 2 * np.pi : resolution * 2j, 0 : np.pi : resolution * 1j]
85         X = radius * np.cos(u) * np.sin(v) + x
86         Y = radius * np.sin(u) * np.sin(v) + y
87         Z = radius * np.cos(v) + z
88         return (X, Y, Z)
89
90
91     def plotly_fig2array(fig):
92         # convert Plotly fig to an array
93         fig_bytes = fig.to_image(format="png")
94         buf = io.BytesIO(fig_bytes)
95         img = Image.open(buf)
96         return np.asarray(img)
97
98
99     class MAI_Trajectory(Trajectory):
100     def __init__(self, env, initial_return, initial_frame=None):
101         self.pos = []
102         self.ori = []
103         self.pts = []
104         self.unobserved_points = []
105         self.cumm_rewards = []
106
107         super().__init__(env, initial_return, initial_frame=initial_frame)
108
109     def add(self, env, action, step_return, agent_info, frame=None):
110         super().add(env, action, step_return, agent_info, frame=frame)
111         frame = env.hills_frame(env.orb["chief"])
112
113         # Here: would be frame.T because we're mapping from absolute coords to
114         # chief frame, so right mul is the same
115         # for orientation, recall that self.ori[k, axis, :] is the k'th
116         # agent's axis, with axis 0 being the heading.
117
118         self.pos.append(copy.copy(env.pos) @ frame)
119         self.ori.append(copy.copy(env.ori) @ frame)
120         self.pts.append(copy.copy(env.pts) @ frame)
121         self.unobserved_points.append(copy.copy(env.unobserved_points))
122         self.cumm_rewards.append(copy.copy(env.cum_rewards))
123
124
125     class MAI_COACH(COACHEnvironment):
126         agent_selection = agents
127
128     def __init__(
129         self,
130         env_creator: callable,
131         parameter_generator: BaseParameterGenerator,
132         agents=dict(),
133         fill_random=True,
134         comm_schedule: Timeline=None,
135         seed=6234235,
136         TrajectoryClass = MAI_Trajectory,
137     ):
138         super().__init__(env_creator,
139             parameter_generator,
140             agents=agents,
141             fill_random=fill_random,
142             comm_schedule=comm_schedule,
143             seed=seed,
144             TrajectoryClass = TrajectoryClass
145         )
146
147     def plot_trajectory_component(
148         self,
149         traj,
150         coas=None,
151         env=None,
152         plan=None,
153         render_mode="plotly",
154         simulate_data=False
155     ):
156         if plan:
157             coas = plan.coas
158
159         players = traj.players

```

```

160 cmap = plt.get_cmap("tab10")
161 event_cmap = plt.get_cmap("Set1")
162 player_to_color = {p: i / 10 for i, p in enumerate(players)}
163
164 game_len = len(traj)
165 num_points = env.num_points
166 chief_perim = env._CHIEF_PERIMETER
167
168 # 3 dim: pos, 4 dim: color
169 xs = np.zeros([game_len, len(players), 7])
170 xs[:, :, :3] = np.array([pos.to(env._OU_DIS).value for pos in traj.pos])
171
172 # Plot Cone Information
173 CONE_FREQ = 5
174 CONE_OPACITY = 0.5
175
176 n_cones = int(np.ceil(game_len/CONE_FREQ))
177 cones = np.zeros([n_cones*len(players), 10])
178
179 # Stack is going to concatenate long the first axis, combine all
180 # players info into one stack
181 cones[:, :3] = np.vstack(xs[:, :CONE_FREQ, :, :3])
182 cones[:, 3:6] = np.vstack(np.array(traj.ori)[:, :CONE_FREQ, :, :])[:, 0, :]
183 pcolors = [cmap(player_to_color[p]) for p in players]
184 cones[:, 6:] = np.repeat(np.array(pcolors), n_cones, axis=0)
185 cones[:, -1] = CONE_OPACITY
186
187 # Seperate into drifts and burns
188 for p_idx, p in enumerate(players):
189     c = cmap(player_to_color[p])
190
191     if p in coas.keys():
192         coa = coas[p]
193     else:
194         coa = COA()
195
196     event_time_left = 0
197
198     for i in range(game_len):
199         if i in coa.timeline.keys():
200             for event in coa.timeline[i]:
201                 # event = coa.timeline[i][action]
202                 event_time_left = event.parameters[0]
203                 print(event.label)
204                 c = event_cmap(event.id)
205                 print(c)
206
207             if event_time_left <= 0:
208                 c = cmap(player_to_color[p])
209
210             xs[i, p_idx, 3:] = c
211             event_time_left -= 1
212
213 ## Get Seen Points
214 pts = traj.pts[0]
215 u_set = traj.unobserved_points[0]
216
217 unseen = np.zeros(pts.shape[0], dtype=bool)
218 unseen[list(u_set)] = True
219
220 seen = pts[~unseen]
221 unseen = pts[unseen]
222
223 traces = []
224 traces.append(plot_orbits(np.vstack(xs)))
225
226 if unseen.shape[0] > 0:
227     traces.append(plot_points(unseen, "#000000"))
228
229 if seen.shape[0] > 0:
230     traces.append(plot_points(seen, "#FFFFFF"))
231
232 # # Draw Chief Sphere
233 (x_pns_surface, y_pns_surface, z_pns_surface) = ms(0, 0, 0, chief_perim)
234 traces.append(
235     go.Surface(
236         x=x_pns_surface,
237         y=y_pns_surface,
238         z=z_pns_surface,
239         opacity=0.5,
240         showscale=False,

```

```

241     )
242 )
243
244 # View Cones:
245 traces.append(
246     go.Cone(
247         x=cones[:, 0],
248         y=cones[:, 1],
249         z=cones[:, 2],
250         u=cones[:, 3],
251         v=cones[:, 4],
252         w=cones[:, 5],
253         opacity=0.1,
254         showscale=False,
255         showlegend=False,
256         sizemode="absolute",
257         sizeref=10,
258         anchor="tip",
259     )
260 )
261
262 fig = go.Figure(data=traces)
263 fig.update_layout(
264     scene_aspectmode="cube",
265     showlegend=False,
266     scene=dict(
267         xaxis=dict(
268             nticks=4,
269             range=[-300, 300],
270         ),
271         yaxis=dict(
272             nticks=4,
273             range=[-300, 300],
274         ),
275         zaxis=dict(
276             nticks=4,
277             range=[-300, 300],
278         ),
279     ),
280     width=700,
281     margin={"l": 40, "b": 40, "t": 10, "r": 0},
282     hovermode="closest",
283 )
284
285 if render_mode == "plotly":
286     return fig
287
288 if render_mode == "rgb_array":
289     return plotly_fig2array(fig)
290
291
292 #####
293 ## Example Usage
294 #####
295
296 if __name__ == "__main__":
297     from examples.MAInspection.env import MultiInspect
298
299     ## Environmental Parameters
300     env_params = {"_OBS_REWARD": .01, "num_deputies": 2}
301
302     # Actor Parameters
303     COACH_params = {
304         "Agents": {
305             "player_0": {
306                 "class_name": "SB_PP0WaypointActor",
307                 "params": {"policy_path":
308                     "/root/coach/examples/MAInspection/waypointer/MAInspect_20240205-234515.zip"} #
309                     If the agent has setup parameters
310             },
311             "player_1": {
312                 "class_name": "SB_PP0WaypointActor",
313                 "params": {"policy_path":
314                     "/root/coach/examples/MAInspection/waypointer/MAInspect_20240205-234515.zip"} #
315                     If the agent has setup parameters
316             },
317         }
318     }
319
320     # Create actors from parameters
321     agent_dict = dict()

```

```

318     if "Agents" in COACH_params.keys():
319         for role, agent in COACH_params["Agents"].items():
320             agent_class = MAI_COACH.agent_selection.Agents[agent["class_name"]]
321             agent_dict[role] = agent_class(role, **agent.get("params", dict()))
322
323     # Example communication scheudle.
324     comms = CommunicationSchedule.repeating(checkin_frequency=10)
325
326     # Wrap parameters in parameter generator
327     parameter_generator = SeededParameterGenerator(23423, env_params)
328
329     def env_creator():
330         return MultiInspect(**env_params)
331
332     env = MAI_COACH(
333         env_creator = MultiInspect,
334         parameter_generator = parameter_generator,
335         comm_schedule = comms,
336         fill_random = True,
337         agents = agent_dict
338     )
339
340     for agt in env.agents.values():
341         print(agt.interface)
342
343     for i in range(2):
344         env.step()
345         traj = env.state.trajectory
346         coas = {role: agt.coa for role, agt in env.agents.items()}
347         fig = env.plot_trajectory_component(traj, coas, env=env.env)
348         fig.show()
349     # %%

```

6.3 examples/MAInspection/Environments/env.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 # %%
15
16 """Multi-Inspection Environment"""
17
18 import argparse # for type hinting
19 import copy
20 import logging
21 from collections.abc import Callable # for type hinting
22 from typing import Any, Optional, TypeVar, Union, cast, NewType
23
24 import gymnasium as gym
25 import imageio
26 import matplotlib # type: ignore[import]
27 import matplotlib.lines as mlines # type: ignore[import]
28 import matplotlib.pyplot as plt # type: ignore[import]
29 import numpy as np
30
31 # from pettingzoo.utils.env import ParallelEnv
32 from matplotlib.backends.backend_agg import (
33     FigureCanvasAgg as FigureCanvas, # type: ignore[import]
34 )
35 from matplotlib.collections import LineCollection
36 from mpl_toolkits.mplot3d.art3d import Line3DCollection # type: ignore[import]
37 from numpy.random import PCG64DXSM, Generator
38 from scipy.spatial.transform import Rotation # type: ignore[import]
39 from scipy.stats import multivariate_normal, ortho_group # type: ignore[import]
40
41 import sys
42
43 from numpy.typing import NDArray
44 FloatArray = NDArray[np.float_]
45 RenderFrame = NDArray[np.uint8]
46 Role = NewType("Role", str)
47
48 from astropy import units as u
49 from poliastro.maneuver import Maneuver
50 from poliastro.bodies import Earth, Mars, Sun
51 from poliastro.twobody import Orbit
52
53 matplotlib.use("agg")
54 logger = logging.getLogger(__name__)
55
56 from pettingzoo import ParallelEnv
57
58 #####
59 ## Factories
60 #####
61 def get_env_class(args: argparse.Namespace) -> Callable[..., Any]:
62     """Returns the class to use, based on input arguments
63
64     Parameters
65     -----
66     args: argparse.Namespace
67         arguments that were passed to the `main()` function
68
69     Returns
70     -----
71     class
72         the class to use in creating env objects
73     """
74     return MultInspect
75
76 def unit(v):
77     if len(v.shape) == 1:
78         v = v.reshape(1,-1)
```

```

79     return v/np.linalg.norm(v,axis=1)[: ,None]
80
81 def proj(v, onto):
82     return onto * np.dot(v, onto) / np.dot(onto, onto)
83
84 def frame(r, v):
85     r = unit(r)
86     v = unit(v)
87     n = np.cross(r,v, axis=1)
88
89     return np.stack([r,v,n],axis=2)
90
91 def parallel_env(**kwargs):
92     return MultInspect(**kwargs)
93
94 class MultInspect(ParallelEnv):
95     """
96     We assume six axis movement:
97         action space: [X-Thrust, Y-Thrust, Z-Thrust,
98                       X-Clockwise Torque, Y-Clockwise Torque, Z-Clockwise Torque]
99
100     Some notes on linear algebra conventions and frames:
101
102     All frames have columns as vectors in the background coordinate system R, so
103     for any frame M,  $M * v$  maps  $v$  to R. To traslate back, a vector in  $w$  in R is mapped to
104      $M^T * w$ .
105
106     A good rule of thumb here is to understand every matrix as having units
107      $xf$ , where  $x$  is standard coords and  $f$  is frame coords. Since frames are orthonormal
108      $M^{-1} = M^T$ . If we have a vector in frame 1 coords and we want it in frame 2
109     coords, we can just pass through the standard coords:  $M_1: v_{f1} \rightarrow v_x$ ,
110      $M_2^T: v_x \rightarrow v_{f2}$ , so  $M_2^T M_1 * v_{f1} = v_{f2}$  is a appropriate transform
111
112     A source for physics: http://control.asu.edu/Classes/MMAE441/Aircraft/441Lecture9.pdf
113
114     We're implementing a Petting Zoo interface, details can be found here:
115     https://pettingzoo.farama.org/api/parallel/#parallelenv
116
117     Attributes
118     -----
119
120     agents: list[AgentID]
121         A list of the names of all current agents, typically integers. May changed as environment
122         progresses
123     num_agents: int
124         The length of the agents list.
125     possible_agents: list[AgentID]
126         A list of all possible_agents the environment could generate. Equivalent to the list of agents
127         in the observation and action spaces. This cannot be changed through play or resetting.
128     max_num_agents: int
129         The length of the possible_agents list.
130     observation_spaces: dict[AgentID, gym.spaces.Space]
131         A dict of the observation spaces of every agent, keyed by name. This cannot be changed through
132         play or resetting.
133     action_spaces: dict[AgentID, gym.spaces.Space]
134         A dict of the action spaces of every agent, keyed by name. This cannot be changed through play
135         or resetting.
136
137     Methods
138     -----
139
140     step(actions: dict[str, ActionType]) tuple[dict[str, ObsType], dict[str, float], dict[str,
141         bool], dict[str, bool], dict[str, dict]]
142         Receives a dictionary of actions keyed by the agent name.
143         Returns the observation dictionary, reward dictionary, terminated dictionary, truncated
144         dictionary and info dictionary, where each dictionary is keyed by the agent.
145     reset(seed: int | None = None, options: dict | None = None) dict[str, ObsType]
146         Resets the environment.
147     seed(seed=None)
148         Reseeds the environment (making it deterministic).
149     render() None | np.ndarray | str | list
150         Displays a rendered frame from the environment, if supported.
151     close()
152         Closes the rendering window.
153     state() ndarray
154         Returns the state.
155     observation_space(agent: str) Space
156         Returns the observation space for given agent.
157     action_space(agent: str) Space

```

```

154     """
155
156     metadata = {"render_modes": ["human", "rgb_array"], "render_fps": 50, "name": "MAInspect"}
157
158
159     def __init__(self, render_mode="rgb_array", **kwargs) -> None:
160         """
161         self.augment(params = kwargs)
162         self.render_mode = render_mode
163
164     def parallel_env(self, **kwargs):
165         return self
166
167     def _setup_gym_env(self, env_params: dict) -> None:
168         ## Setup parameters. All of this may be overwritten by the
169         ## params dictionary if the appropriate key exists.
170         ## The list below provides the default parameters
171
172         self.verbose: bool = False
173         self.six_axis: bool = True
174         self.num_deputies: int = 3
175         self.max_episode_steps: int = 500
176         self.num_points: int = 20
177
178         self.MAXTIME = 1 << u.h
179         self.TIME_PER_STEP = 1 << u.min
180
181         # Computability
182         # This disable some agents terminating before others.
183         self._SB_SAFTY_MODE = True
184
185         # Game Mode:
186         self._TRAIN_WAYPOINTER = True
187         self._WAYPOINT_ARRIVAL_PROX = 10 << u.m
188         self._WAYPOINT_ARRIVE_REWARD = 1
189
190         # Initial Conditions
191         self.INIT_ALT = 700 << u.km
192         self.INIT_ALT_OFFSET = 50 << u.m
193         self.RAAN = 0 << u.deg ## Angle Around the Circular Orbit
194         self.ARG_LAT = 0 << u.deg ## Latitude updown
195         self.INC = 0 << u.deg ## Direction of orbit
196         self._STARTING_DISTANCE_FROM_CHIEF = 150 << u.m
197
198         self.offset_angle = (1/10)*(self._STARTING_DISTANCE_FROM_CHIEF /
199                                     self.INIT_ALT).decompose().value << u.rad
200
201         # Action Frame
202         self.ACTION_FRAME = "Hills" # Options should be Hills and Orientation
203         self.OBSERVATION_FRAME = "Hills" # Options should be Hills and Orientation
204         self._OU_DIS = u.m
205         self._OU_TIM = u.s
206         self._OU_VEL = self._OU_DIS/self._OU_TIM
207
208         self._CHIEF_PERIMETER: float = 50 << u.m
209         self._DEPUTY_PERIMETER: float = 150 << u.m
210         self._DEPUTY_MASS: float = 1 << u.kg
211         self._DEPUTY_RADIUS: float = 1 << u.m
212         self._DEPUTY_THRUST_COEFF: float = 0.01
213
214         self._SIM_OUTER_PARAMETER: float = 200 << u.m #
215         self._MAX_OUTER_PERIMETER: float = 300 << u.m # If you leave here, you truncate
216         self._STARTING_VEL_NORM: float = 0 << u.m/u.s # 0.001
217
218         self._TIME_PER_STEP: float = 90 << u.s
219         self._DELTA_V_PER_THRUST: float = 0.1 << u.m/u.s
220         self._DELTA_T: float = 90 << u.s
221         self._TAU_per_THRUST: float = 0.1
222         self._USE_ANGULAR_MOMENTUM = False
223
224         self._OBS_REWARD: float = 0.02 # 20
225         self._REWARD_FOR_SEEING_ALL_POINTS: float = 1 # 200
226         self._CRASH_REWARD: float = 0
227         self._REWARD_PER_STEP: float = 0.0 # -.1 # Reward per tick
228         self._REWARD_FOR_LEAVING_PARAMETER: float = 0
229         self._REWARD_FOR_LEAVING_OUTER_PERIMETER: float = 0
230
231         self._SOLID_CHIEF: bool = True
232         self._MAX_BURN: float = 10
233
234         self._REW_COV_MTRX = 300**2

```

```

234     self._PROX_RWD_SCALE = 100000
235
236     # Visualization
237     self._VIS_NUM_CONE_LINES = 8
238     self._VIS_CONE_LEN = 40 << u.m
239
240     # Vision Cone:
241     self._MIN_VISION: float = 0 << u.m
242     self._MAX_VISION: float = np.inf << u.m
243     self._VISION_ARC: float = np.pi / 8 << u.rad
244
245     # Apply parameters from input
246     self._apply_parameters(params=env_params)
247     if hasattr(env_params, "args"):
248         self.master_seed: int = env_params.args.master_seed
249     else:
250         self.master_seed: int = 487924
251     self.seed(seed=self.master_seed)
252
253
254     if self._TRAIN_WAYPOINTER:
255         self.num_points = self.num_deputies
256
257     # Check parameters from input
258     assert self._MIN_VISION >= 0, "_MIN_VISION must be >= 0"
259     assert self._MAX_VISION > self._MIN_VISION, "_MAX_VISION must be > _MIN_VISION"
260     assert (self._VISION_ARC >= 0) and (
261         self._VISION_ARC <= np.pi << u.rad
262     ), "_VISION_ARC must be [0, pi]"
263
264     ## Setup actual Gym Env based on the above
265     self.agents: list[Role] = [Role(f"player_{i}") for i in range(self.num_deputies)]
266
267     # Agent Velocity, Other Dep. Rel Position and Vel, Point Rel Position
268     self._VISION: dict[Role, list[float]] = {
269         role: [self._VISION_ARC, self._MIN_VISION, self._MAX_VISION]
270         for role in self.agents
271     }
272
273     in_size: int = (
274         6 + 6 * (self.num_deputies - 1) + 3 * self.num_points + self.num_points
275     )
276
277     if self._TRAIN_WAYPOINTER:
278         raw_low_values: list[float] = [
279             -self._MAX_OUTER_PERIMETER.value * np.ones(3), # Chief Position
280             -self._MAX_OUTER_PERIMETER.value
281             * np.ones(3 * self.num_deputies), # Deputy velocity
282             -2
283             * self._MAX_OUTER_PERIMETER.value
284             * np.ones(3 * (self.num_deputies - 1)), # Dep. Positions, except me
285             -2
286             * self._MAX_OUTER_PERIMETER.value
287             * np.ones(3), # Single Waypoint,
288         ]
289
290         raw_high_values: list[float] = [
291             self._MAX_OUTER_PERIMETER.value * np.ones(3), # Chief Position
292             self._MAX_OUTER_PERIMETER.value
293             * np.ones(3 * self.num_deputies), # Deputy velocity
294             2
295             * self._MAX_OUTER_PERIMETER.value
296             * np.ones(3 * (self.num_deputies - 1)), # Dep. Positions, except me
297             2
298             * self._MAX_OUTER_PERIMETER.value
299             * np.ones(3), # Single Waypoint,
300         ]
301     else:
302         raw_low_values: list[float] = [
303             -self._MAX_OUTER_PERIMETER.value * np.ones(3), # Chief Position
304             -self._MAX_OUTER_PERIMETER.value
305             * np.ones(3 * self.num_deputies), # Deputy velocity
306             -2
307             * self._MAX_OUTER_PERIMETER.value
308             * np.ones(3 * (self.num_deputies - 1)), # Dep. Positions, except me
309             -2
310             * self._MAX_OUTER_PERIMETER.value
311             * np.ones(3 * (self.num_points)), # Point Pos,
312             np.zeros(self.num_points),
313         ]
314

```



```

315         raw_high_values: list[float] = [
316             self._MAX_OUTER_PERIMETER.value * np.ones(3), # Chief Position
317             self._MAX_OUTER_PERIMETER.value
318             * np.ones(3 * self.num_deputies), # Deputy velocity
319             2
320             * self._MAX_OUTER_PERIMETER.value
321             * np.ones(3 * (self.num_deputies - 1)), # Dep. Positions, except me
322             2
323             * self._MAX_OUTER_PERIMETER.value
324             * np.ones(3 * (self.num_points)), # Point Pos,
325             np.ones(self.num_points),
326         ]
327
328         low = np.concatenate(raw_low_values, dtype=np.float32)
329         high = np.concatenate(raw_high_values, dtype=np.float32)
330
331         obs_space = gym.spaces.Box(low=low, high=high, dtype=np.float32)
332
333         if self.six_axis:
334             low = np.array([-10, -10, -10, -np.pi, -np.pi, -np.pi], dtype=np.float32)
335             act_space = gym.spaces.Box(low=low, high=-low, dtype=np.float32)
336         else:
337             # Thrust, rotate Left/Right, rotate UP/Down
338             act_space = gym.spaces.Box(
339                 low=-np.ones(3, dtype=np.float32),
340                 high=np.ones(3, dtype=np.float32),
341                 dtype=np.float32,
342             )
343
344         self.possible_agents = self.agents
345
346         self.observation_spaces = {role: obs_space for role in self.possible_agents}
347         self.action_spaces = {role: act_space for role in self.possible_agents}
348
349         self.ori = np.zeros([self.num_deputies, 3, 3]) # Depute Orientation relative to absolute
350                 earth frame
351         self.rot = np.zeros([self.num_deputies, 3]) # Depute Angular Momentum
352
353         self.pos = np.zeros([self.num_deputies, 3]) # Depute Position Holder
354         self.vel = np.zeros([self.num_deputies, 3]) # Depute Velocity Holder
355
356         self.chief_frame = np.zeros([3, 3]) # Current frame for the cheif relative to
357                 absolute earth frame
358         self.frames = np.zeros([self.num_deputies, 3, 3]) # Current frame for the deputies relative
359                 to absolute earth frame
360         self.pts = np.zeros([self.num_deputies, 3]) # Chief Inspection Points
361         self.nor = np.zeros([self.num_deputies, 3]) # Chief Inspection Normals
362
363         self.sim_steps = 0
364
365         self.unobserved_points = set(range(self.num_points))
366
367         self.render_path: Optional[str] = None
368         self.frame_store: list[RenderFrame] = []
369         self.cum_rewards = np.zeros(self.num_deputies)
370
371         def _apply_parameters(self, params: dict) -> None:
372             for k, v in params.items():
373                 self.__dict__[k] = copy.copy(v)
374
375         def augment(self, params: dict) -> None:
376             self._setup_gym_env(env_params=params)
377
378         def seed(self, seed: int) -> None:
379             self.np_random = Generator(PCG64DXSM(seed=seed))
380
381         def observation_space(self, agent: Role) -> gym.spaces.Box:
382             return self.observation_spaces[agent]
383
384         def action_space(self, agent: Role) -> gym.spaces.Box:
385             return self.action_spaces[agent]
386
387         ## Reset
388         def reset(
389             self,
390             *,
391             seed: Optional[int] = None,
392             options: Optional[dict[Any, Any]] = None,
393             render_path: Optional[str] = None,
394         ) -> tuple[dict[Role, Any], dict[Role, dict[str, Any]]]:
395             """ """

```

```

393         self.sim_steps = 0
394
395         # Reset Seed
396         local_seed = seed if (seed is not None) else self.master_seed
397         self.seed(seed=local_seed)
398         local_og = ortho_group(dim=3, seed=self.np_random)
399
400         # Create Orbits for centers of mass
401         self.orb = dict()
402         self.orb_hist = []
403
404         # Construct Chief
405         self.orb[f"chief"] = Orbit.circular(Earth, alt=self.INIT_ALT)
406         if self._TRAIN_WAYPOINTER:
407             self.pts = self._random_sphere(self.num_points, radius=self._DEPUTY_PERIMETER)
408         else:
409             self.pts = self._random_sphere(self.num_points, radius=self._CHIEF_PERIMETER)
410
411         self.chief_angular_momentum = None
412         # self.chief_angular_momentum = Rotation.from_euler("xyz", [1,0,0]).as_matrix()
413
414         # Construct Deputies
415         self._active = np.array([role in self.agents for role in self.possible_agents])
416
417         # Deputy Orbits
418         for i in range(self.num_deputies):
419             self.orb[f"player_{i}"] = Orbit.circular(Earth, alt=self.INIT_ALT + i*self.INIT_ALT_OFFSET,
420                                                         raan = self.offset_angle)
421
422         self.ori = np.stack([local_og.rvs() for i in range(self.num_deputies)])
423         self.rot = self._random_sphere(self.num_deputies, radius=self._CHIEF_PERIMETER)
424
425         # Make Initial Observations
426
427         # Compute rewards
428         if self._TRAIN_WAYPOINTER:
429             obs = self._make_waypoint_observations()
430         else:
431             obs = self._make_observations()
432
433         self.unobserved_points = set(range(self.num_points))
434         self.truncated = {Role(f"player_{i}"): False for i in range(self.num_deputies)}
435
436         # Initial relative positions and velocities
437         self.pos = np.stack([self.orb[role].r-self.orb["chief"].r for role in self.possible_agents])
438         self.vel = np.stack([self.orb[role].v-self.orb["chief"].v for role in self.possible_agents])
439
440         # Setup Reward Structure for waypoints
441         self.pt_prox = []
442         for i in range(self.num_points):
443             self.pt_prox.append(multivariate_normal(mean=self.pts[i], cov=self._REW_COV_MTRX))
444
445         self.cum_rewards = np.zeros(self.num_deputies)
446
447         # Setup Longterm Rendering
448         self.render_path = render_path
449         if render_path:
450             frame = self.render()
451             assert frame is not None
452             self.frame_store = [frame]
453
454         self.render_data = dict()
455
456         return obs, {agt: dict() for agt in self.possible_agents}
457
458 def step(
459     self, action: dict[str, FloatArray]
460 ) -> tuple[
461     dict[Role, Any],
462     dict[Role, float], # reward
463     dict[Role, bool], # terminated
464     dict[Role, bool], # truncated
465     dict[Role, dict[str, Any]], # info
466 ]:
467     """ """
468     # truncated = self.truncated
469     self.terminated_this_step = False
470
471     # Propagate Motion In Time
472     self._propagate_objects(action)

```

```

473 self.pos = np.stack([self.orb[role].r-self.orb["chief"].r for role in self.possible_agents])
474 self.vel = np.stack([self.orb[role].v-self.orb["chief"].v for role in self.possible_agents])
475
476 # Compute Which Points Seen
477 just_seen = self._detect_points()
478
479 # Compute rewards
480 if self._TRAIN_WAYPOINTER:
481     self._compute_waypoint_reward(just_seen)
482     obs = self._make_waypoint_observations()
483 else:
484     self._compute_reward(just_seen)
485     obs = self._make_observations()
486
487 # Check if all of the deputies have truncated.
488 if all(self.truncated.values()):
489     terminated = True
490
491 if self.render_path:
492     frame = self.render()
493     assert frame is not None
494     self.frame_store.append(frame)
495
496     if terminated:
497         imageio.mimwrite(
498             uri=self.render_path,
499             ims=self.frame_store,
500             fps=int(60 / 10), # type: ignore[arg-type]
501             loop=0,
502         )
503
504 self.cum_rewards += self.sra
505
506 self.obs = obs
507 self.reward = {role: rwd[0] for role, rwd in self.step_rewards.items()}
508
509 self.info: dict[Role, dict[str, Any]] = {
510     agt: dict() for agt in self.possible_agents
511 }
512
513 self.terminated = {agt: self.terminated_this_step for agt in self.possible_agents}
514
515 # Remove Terminated and Truncated agents from available agent lists
516 if self._SB_SAFTY_MODE:
517     self.agents = self.possible_agents
518 else:
519     self.agents = []
520     for role in self.possible_agents:
521         if not (self.terminated[role] or self.truncated[role]):
522             self.agents.append(role)
523
524 # Set the internal active players array
525 self._active = np.array([role in self.agents for role in self.possible_agents])
526
527 return (self.obs, self.reward, self.terminated, self.truncated, self.info)
528
529
530
531 def _compute_reward(self, just_seen: list[list[int]]) -> float:
532     self.sra = np.zeros(self.num_deputies) # Step Reward Array
533     self.step_rewards = {role: self.sra[i:i+1] for i, role in enumerate(self.agents)} # Slices are
534         pointers to the reward array
535
536     self._insepct_reward(just_seen)
537     self._chief_prox_reward()
538     self._game_length_exceed()
539     self._seeing_all_points()
540     self._leaving_outer_perimeter()
541
542 def _compute_waypoint_reward(self, just_seen: list[list[int]]) -> float:
543     self.sra = np.zeros(self.num_deputies) # Step Reward Array
544     self.step_rewards = {role: self.sra[i:i+1] for i, role in enumerate(self.agents)} # Slices are
545         pointers to the reward array
546
547     self._go_towards_waypoint()
548     self._game_length_exceed()
549     self._leaving_outer_perimeter()
550     self._chief_prox_reward()
551

```

```

552 def _go_towards_waypoint(self):
553     # We're going to change the reward structure to include waypointing
554     for i, role in enumerate(self.agents):
555         self.step_rewards[role] += self.pt_prox[i].pdf(self.pos[i]) * self._PROX_RWD_SCALE
556         if np.linalg.norm(self.pos[i] - self.pts[i]) < self._WAYPOINT_ARRIVAL_PROX:
557             self.step_rewards[role] += self._WAYPOINT_ARRIVE_REWARD
558             self.terminated_this_step = True
559
560
561
562 def _seeing_all_points(self):
563     if len(self.unobserved_points) == 0:
564         R = (self.max_episode_steps - self.sim_steps) / self.max_episode_steps
565         self.sra += R * self._REWARD_FOR_SEEING_ALL_POINTS
566         self.terminated_this_step = True
567
568
569
570 def _leaving_outer_perimeter(self):
571     I = np.where(np.linalg.norm(self.pos, axis=1) > self._MAX_OUTER_PERIMETER)[0]
572
573     # print(np.linalg.norm(self.pos, axis=1) > self._MAX_OUTER_PERIMETER)
574     # print(I, self.possible_agents, self.pos)
575     for i in I:
576         role = Role(f"player_{i}")
577         if not self.truncated[role]:
578             self.truncated[role] = True
579             self.step_rewards[role] += self._REWARD_FOR_LEAVING_OUTER_PERIMETER
580
581
582
583 def _game_length_exceed(self):
584     self.sim_steps += 1
585     if self.sim_steps > self.max_episode_steps:
586         ## Technically we were truncated
587         self.truncated = {
588             Role(f"player_{i}"): True for i in range(self.num_deputies)
589         }
590         self.terminated_this_step = True
591
592
593
594 def _insepct_reward(self, just_seen):
595     """ """
596     for i, role in enumerate(self.possible_agents):
597         seen_points = just_seen[i]
598         # Initialize reward
599         # Start with the survival reward
600         reward: float = self._REWARD_PER_STEP
601
602         # Check if seen points are new
603         new_pts: set = self.unobserved_points.intersection(seen_points)
604
605         # Check if new points were seen
606         if new_pts:
607             # Reward for seeing points for the first time
608             reward += len(new_pts) * self._OBS_REWARD
609             # Update unseen points
610             self.unobserved_points = self.unobserved_points.difference(new_pts)
611
612         self.step_rewards[role] += reward
613
614
615
616 def _chief_prox_reward(self):
617     # Are we active and outside the safe zone?
618     outside_safe_zone = self._active & (np.linalg.norm(self.pos, axis=1) > self._SIM_OUTER_PARAMETER)
619     # Are we active and inside the chief?
620     inside_chief = self._active & (np.linalg.norm(self.pos, axis=1) < self._CHIEF_PERIMETER)
621
622     # Adjust reward
623     self.sra[outside_safe_zone] += self._REWARD_FOR_LEAVING_PARAMETER
624     self.sra[inside_chief] += self._CRASH_REWARD
625
626     # Kill agents inside chief
627     self._active[inside_chief] = False
628
629     for i in np.where(inside_chief)[0]:
630         self.terminated_this_step = True
631
632

```

```

633 def _random_sphere(self, n_points: int, radius: float) -> FloatArray:
634     """ """
635     ## Generate random points on a sphere
636     pts: FloatArray = -1 + 2 * self.np_random.normal(size=[n_points, 3])
637     pts = pts / np.linalg.norm(pts, axis=1).reshape(-1, 1)
638     pts = pts * radius
639
640     return pts
641
642 # Physics goes here
643 def _propagate_objects(self, actions):
644     # Poliastro assumes earth centered coordinates. It fixes an x,y, and z. So
645     # vectors in poliastro objects are in terms of of an absolute coordinate system.
646
647     # Translate actions into action frame and apply impulses
648
649     for player, (role, act) in enumerate(actions.items()):
650         # Compute Velocity Change
651         dv = act[:3].reshape(-1,1) * self._DELTA_V_PER_THRUST
652         if self.ACTION_FRAME == "Hills":
653             frame = self.hills_frame(self.orb[role])
654         else:
655             frame = self.ori[role]
656
657         imp = Maneuver.impulse((frame @ dv).reshape(-1))
658         self.orb[role] = self.orb[role].apply_maneuver(imp)
659
660         # Apply rotations
661         # Sanity Check: if frame is ori, than an action of [1,0,0]
662         # fixes ori[:,0] and rotates ori[:,1] and ori[:,2]
663         # Here, R has units ff, mapping from frame coords to frame coords
664         # with left multiplaction assumed for standard clockwise rotation
665
666         tau = self._TAU_per_THRUST * act[-3:]
667
668         if self._USE_ANGULAR_MOMENTUM:
669             self.rot[player] += tau
670             tau = self.rot
671
672         R = Rotation.from_euler("xyz", tau)
673         T = frame @ R.as_matrix() @ frame.T # x columns to frame, rotate, then back to x
674
675         self.ori[player, :, :] = T @ self.ori[player, :, :]
676
677     # Rotate Chief Points:
678     if self.chief_angular_momentum is not None:
679         R = self.chief_angular_momentum
680         self.pts = R @ self.pts
681
682     # Propagate Orbits
683     for role, orb in self.orb.items():
684         self.orb[role] = orb.propagate(self._TIME_PER_STEP)
685
686     # for role in self.possible_agents:
687     #     print(role, self.orb[role].r, np.linalg.norm(self.orb[role].r - self.orb["chief"].r))
688
689 def _detect_points(self):
690     just_seen = []
691     for player, role in enumerate(self.possible_agents):
692         # Position relvative to chief
693         pos = self.orb[role].r - self.orb["chief"].r
694
695         # Detect Points:
696         # This handles occlusion by the chief
697         # Basically, what points are on the same side of the chief as you
698         seeable_points = np.where(
699             (self.pts * (pos - self.pts)).sum(axis=1) > 0
700         )[0]
701
702         #####
703         # Spherical Vision
704         #####
705         # point holder
706         seen_points: list[int] = []
707
708         # loop over possible points
709         for point in seeable_points:
710             # Cone References
711             # cone:
712             https://stackoverflow.com/questions/12826117/how-can-i-detect-if-a-point-is-inside-a-cone-or-not-in-3

```

```

713         # We're going to not use the cone, and switch to using a shell
714         # The initial distance calc though is fine, and easy
715         #
716         # Shell References
717         # This is a pain in the a$$ - I hate math vs physics
718         # Used all of these references to generate a consensus algorithm
719         # https://en.wikipedia.org/wiki/Spherical_coordinate_system
720         # This one doesn't fully work
721         # https://en.wikipedia.org/wiki/Atan2
722         # This explains why atan2 vs atan
723         # https://mathworld.wolfram.com/SphericalCoordinates.html
724         #
725         # https://stackoverflow.com/questions/4116658/faster-numpy-cartesian-to-spherical-coordinate-conversion
726         # THIS IS MY FAVORITE ONE - uses physics notation
727         #
728         # https://math.libretexts.org/Bookshelves/Calculus/Calculus_(OpenStax)/12%3A_Vectors_in_Space/12.07%3A_
729         # This supports the stackoverflow, just in math notation
730
731         # Position relative to agent
732         v = self.pts[point, :] - pos
733
734         # Distance relative to agent
735         # this is the same as the distance in the deputy frame, but
736         # doesn't require the transformation
737         p_dist = np.linalg.norm(v)
738
739         # Check if distance is within shell
740         # Ignoring angles at the moment
741         # Less than max, more than min
742         # I assume we're usually too far away
743         if (p_dist < self._VISION[role][2]) and (
744             p_dist >= self._VISION[role][1]
745         ):
746             # Orient point in deputy frame
747             pdf = self.ori[player].T @ v
748
749             # theta and phi
750             # This is physics notation
751             # theta = polar/zenith angle, [0, pi], z-axis is 0
752             # phi = azimuth angle, (-pi, pi], x-axis is 0
753             theta = np.arccos(pdf[2] / p_dist)
754             phi = np.arctan2(pdf[1], pdf[0])
755
756             # theta needs to measure from x-axis, not z-axis
757             # calculate the complementary angle
758             # theta = [pi/2, -pi/2], x-axis is 0
759             theta = (np.pi / 2 << u.rad) - theta
760
761             # Check if point is within cone
762             # Assume circular directions, so abs() it
763             # NOTE: We could implement different ranges for theta and phi
764             # This would give a non-circular viewing angle
765             if (np.abs(phi) < self._VISION[role][0]) and (
766                 np.abs(theta) < self._VISION[role][0]
767             ):
768                 # you can see it!
769                 seen_points.append(point)
770
771             just_seen.append(seen_points)
772             if self.verbose:
773                 print(f"{player} seeable: {seeable_points}, seen: {seen_points}")
774
775         return just_seen
776
777     def accel(self, t0, state, k, rate=1e-5):
778         """Constant acceleration aligned with the velocity. """
779         v_vec = state[3:]
780         #v_vec = state[:3]
781         norm_v = (v_vec * v_vec).sum() ** 0.5
782         return -rate * v_vec / norm_v
783
784     def f(self, t0, u_, k):
785         # t_0: time to evaluate at
786         # u_ = [x,y,z,vx,vy,vz] in earth coords, rather annoyingly unitless.
787         # Assumed units (I've tested this) are km and km/s
788
789         # U.append(u_)
790         du_kep = func_twobody(t0, u_, k)
791         #ax, ay, az = self.accel(t0, u_, k, rate=1e-5)
792         ax, ay, az = self.acc
793         du_ad = np.array([0, 0, 0, ax, ay, az])

```

```

792         return du_kep + du_ad
793
794     def hills_frame(self, orb):
795         r = orb.r
796         v = orb.v
797         v_p = v - proj(v, onto=r)
798         return frame(r, v_p)[0].decompose().value
799
800     def _make_observations(self, obs_frame=None) -> dict[Role, FloatArray]:
801         """ """
802         # Return dictionary
803         # Play role will be keys, observations will be values
804         obs = dict()
805
806         if obs_frame is None:
807             obs_frame = self.OBSERVATION_FRAME
808
809         # Setup point mask
810         # hide points that have been observed
811         observedPoints = list(set(range(self.num_points)) - self.unobserved_points)
812         pt_mask = np.ones(self.num_points)
813         pt_mask[observedPoints] = 0
814
815         poss = np.stack([self.orb[role].r for role in self.possible_agents]).T.to(self._OU_DIS).value
816         vels = np.stack([self.orb[role].v for role in self.possible_agents]).T.to(self._OU_VEL).value
817         pos_c = self.orb["chief"].r.reshape(-1,1).to(self._OU_DIS).value
818         vel_c = self.orb["chief"].v.reshape(-1,1).to(self._OU_VEL).value
819         pos_p = self.pts.T.to(self._OU_DIS).value
820
821         # loop over all deputies
822         for i, role in enumerate(self.possible_agents):
823             # Recall: Moving from absolute coords into a frame is left multiplication
824             # by the transpose
825
826             # Velocities
827             # Velocities of all deputies in my frame
828             if obs_frame == "Hills":
829                 frame = self.hills_frame(self.orb[role])
830             else:
831                 frame = self.ori[i]
832
833             vel_in_frame = frame.T @ (vels - vels[:, [i]])
834             chief_vel_in_frame = frame.T @ (vel_c - vels[:, [i]])
835
836             # Relative velocities of all other deputies in my frame
837             rel_vel = np.delete(arr=vel_in_frame, obj=i, axis=1)
838
839             # Relative positions of all other deputies in my frame
840             rel_pos = (
841                 frame.T @ np.delete(arr=pos - pos[:, [i]], obj=i, axis=1)
842             )
843
844             dchief = pos_c - pos[:, [i]]
845
846             # Relative positions of all points in my frame
847             rel_pts = frame.T @ (pos_p + dchief)
848
849             # Relative position of chief in my
850             rel_chief = frame.T @ dchief
851
852             # Put the observation space together
853             tmp = np.concatenate(
854                 [
855                     rel_chief,
856                     chief_vel_in_frame,
857                     rel_vel,
858                     rel_pos,
859                     rel_pts,
860                 ], axis=1
861             )
862
863             # Store observation space under appropriate role
864             obs[Role(f"player_{i}")] = np.concatenate([tmp.reshape(-1), pt_mask])
865
866         return obs
867
868     def _make_waypoint_observations(self, obs_frame=None) -> dict[Role, FloatArray]:
869         """ """
870         # Return dictionary
871         # Play role will be keys, observations will be values
872         obs = dict()

```

```

873
874     if obs_frame is None:
875         obs_frame = self.OBSERVATION_FRAME
876
877     # Setup point mask
878     # hide points that have been observed
879     observedPoints = list(set(range(self.num_points)) - self.unobserved_points)
880     pt_mask = np.ones(self.num_points)
881     pt_mask[observedPoints] = 0
882
883     poss = np.stack([self.orb[role].r for role in self.possible_agents]).T.to(self._OU_DIS).value
884     vels = np.stack([self.orb[role].v for role in self.possible_agents]).T.to(self._OU_VEL).value
885     pos_c = self.orb["chief"].r.reshape(-1,1).to(self._OU_DIS).value
886     vel_c = self.orb["chief"].v.reshape(-1,1).to(self._OU_VEL).value
887     pos_p = self.pts.T.to(self._OU_DIS).value
888
889
890     # loop over all deputies
891     for i, role in enumerate(self.possible_agents):
892         # Recall: Moving from absolute coords into a frame is left multiplication
893         # by the transpose
894
895         # Velocities
896         # Velocities of all deputies in my frame
897         if obs_frame == "Hills":
898             frame = self.hills_frame(self.orb[role])
899         else:
900             frame = self.ori[i]
901
902         vel_in_frame = frame.T @ (vels - vels[:,[i]])
903         chief_vel_in_frame = frame.T @ (vel_c - vels[:,[i]])
904
905         # Relative velocities of all other deputies in my frame
906         rel_vel = np.delete(arr=vel_in_frame, obj=i, axis=1)
907
908         # Relative positions of all other deputies in my frame
909         rel_pos = (
910             frame.T @ np.delete(arr=pos - pos[:,[i]], obj=i, axis=1)
911         )
912
913         dchief = pos_c - pos[:,[i]]
914
915         # print("dchief", dchief)
916         # print(self.pts.T.to(self._OU_DIS).value + dchief)
917         # Relative positions of all points in my frame
918         rel_pts = frame.T @ (pos_p[:,[i]] + dchief)
919
920         # Relative position of chief in my
921         rel_chief = frame.T @ dchief
922
923         # Put the observation space together
924         tmp = np.concatenate(
925             [
926                 rel_chief,
927                 chief_vel_in_frame,
928                 rel_vel,
929                 rel_pos,
930                 rel_pts,
931             ], axis=1
932         )
933
934         # Store observation space under appropriate role
935         obs[Role(f"player_{i}")] = tmp.reshape(-1)
936
937     return obs
938
939
940
941 ## Visualizations
942 def render(
943     self,
944     render_mode: str = "rgb_array",
945     vision: Optional[int] = 0, # player number, but not their role, just the number
946     rotate: bool = False,
947     elev: float = 45, # I think it's degrees?
948     close: bool = False,
949     **kwargs: Any,
950 ) -> Union[RenderFrame, None]:
951     """ """
952
953     if close:

```



```

954         self.close()
955         return None
956
957     # init figure and canvas
958     fig = plt.figure(figsize=[20,5])
959     canvas = FigureCanvas(fig)
960
961     poss = np.stack([self.orb[role].r-self.orb["chief"].r for role in
962                     self.possible_agents]).T.to(self._OU_DIS).value
963     vels = np.stack([self.orb[role].v-self.orb["chief"].v for role in
964                     self.possible_agents]).T.to(self._OU_VEL).value
965
966     #####
967     # Title
968     #####
969     # plot scores as title
970     score_title = " ".join(
971         ["Player {i}: {self.cum_rewards[i]:.3f}" for i in range(self.num_deputies)]
972     )
973     fig.suptitle(
974         t=score_title, horizontalalignment="center", verticalalignment="center"
975     )
976
977     #####
978     # Setup Plot Area
979     #####
980     # Plot just the chief, or the chief and one agent-centric view?
981     if vision is not None:
982         ax = fig.add_subplot(141, projection="3d")
983     else:
984         ax = plt.axes(projection="3d")
985
986     #####
987     # First Sub Area
988     #####
989     # Orientation
990     deg = 45
991     if (vision is not None) and rotate:
992         x, y = poss[vision, 0:2] / np.linalg.norm(poss[vision, 0:2])
993         deg = np.degrees(np.arctan2(y, x))
994
995     ax.view_init(elev=elev, azim=deg)
996
997     # limits
998     # ax.set_xlim([-300, 300])
999     # ax.set_ylim([-300, 300])
1000     # ax.set_zlim([-300, 300])
1001
1002     #####
1003     # Deputies
1004     #####
1005     # I think all of this just plots deputies - JB
1006
1007     # Deputy body
1008     for i in range(self.num_deputies):
1009         x, y, z = poss[:, [i]]
1010         ax.plot3D(x, y, z, marker="^", linewidth=0, label=f"player_{i}")
1011
1012     # Velocity
1013     # Plot line collection: LC = np.array([N,S,D]), num lines, num of points
1014     # per line, dim of space, so LC[2,0,:] is point 0 on line 2
1015
1016     TPS = (self.TIME_PER_STEP * self._OU_VEL/self._OU_DIS).decompose().value
1017
1018     segs = np.stack([poss.T, poss.T + TPS * vels.T], axis=1)
1019     line_segments = Line3DCollection(
1020         segs, linestyle="solid", label="Velocity", color="k"
1021     )
1022     ax.add_collection(line_segments)
1023
1024     # Orientation
1025     colors = ["r", "g", "b"]
1026     labels = ["Roll/Heading", "Yaw", "Pitch"]
1027     for axis in range(len(labels)):
1028         segs = np.stack([poss.T, poss.T + 30 * self.ori[:, axis, :]], axis=1)
1029         line_segments = Line3DCollection(
1030             segs, linestyle="solid", color=colors[axis], label=labels[axis]
1031         )
1032         ax.add_collection(line_segments)

```

```

1033 #####
1034 # Points on Chief
1035 #####
1036 # Shift for plotting
1037 view_vec = 100 * np.ones(3) / np.sqrt(3)
1038
1039 # Unobserved points
1040 if len(self.unobserved_points) > 0:
1041     # get positions of points
1042     unobserved_pos = self.pts[list(self.unobserved_points), :]
1043     # calc vector for point size
1044     s = np.linalg.norm(unobserved_pos.to(self._OU_DIS).value + view_vec, axis=1) - 40
1045     # get in euclidean space
1046     x, y, z = unobserved_pos.T
1047     # plot
1048     ax.scatter(x, y, z, label="Unseen", s=s, color="m")
1049 else:
1050     # plot for legend
1051     # 60: comes from s, when unobserved_points is the empty set
1052     ax.scatter([], [], [], label="Unseen", s=60, color="m")
1053
1054 # Observed points
1055 observed_pts = set(range(self.num_points)) - self.unobserved_points
1056 if len(observed_pts) > 0:
1057     # get positions of points
1058     observed_pos = self.pts[list(observed_pts), :]
1059     # calc vector for point size
1060     s = np.linalg.norm(observed_pos.to(self._OU_DIS).value + view_vec, axis=1) - 40
1061     # get in euclidean space
1062     x, y, z = observed_pos.T
1063     # plot
1064     ax.scatter(x, y, z, color="c", s=s, label="Seen")
1065 else:
1066     # plot for legend
1067     # 60: comes from s, when observed_points is the empty set
1068     ax.scatter([], [], [], color="c", s=60, label="Seen")
1069
1070 #####
1071 # Legend
1072 #####
1073 # get legend handles
1074 # handles, _ = ax.get_legend_handles_labels()
1075
1076 # Style
1077 fig.legend(loc="lower center", ncol=3, fancybox=True) # , handles=handles
1078
1079 #####
1080 # Chief Sphere
1081 #####
1082 u, v = np.mgrid[0 : 2 * np.pi : 20j, 0 : np.pi : 10j] # type: ignore[misc]
1083 x = self._CHIEF_PERIMETER * np.cos(u) * np.sin(v)
1084 y = self._CHIEF_PERIMETER * np.sin(u) * np.sin(v)
1085 z = self._CHIEF_PERIMETER * np.cos(v)
1086 ax.plot_wireframe(x, y, z, color="k", alpha=0.2)
1087
1088 mins = np.min(poss.T, axis=0) - 10
1089 maxes = np.max(poss.T, axis=0) + 10
1090
1091 widths = []
1092 centers = []
1093 for i, s in enumerate(['x', 'y', 'z']):
1094     lim = getattr(ax, f"get_{s}lim")()
1095     lim = [
1096         min(max(mins[i], -300), -self._CHIEF_PERIMETER.to(self._OU_DIS).value+10),
1097         max(min(maxes[i], 300), self._CHIEF_PERIMETER.to(self._OU_DIS).value+10)
1098     ]
1099     widths.append(lim[1]-lim[0])
1100     centers.append((lim[1]+lim[0])/2)
1101
1102 w = max(widths)/2
1103 for i, s in enumerate(['x', 'y', 'z']):
1104     getattr(ax, f"set_{s}lim")([centers[i] - w, centers[i] + w])
1105
1106 #####
1107 # Render Agent View
1108 #####
1109 if vision is not None:
1110     ax2 = fig.add_subplot(142, projection="3d")
1111     all_obs = self._make_observations(obs_frame="Orientation")
1112     role = cast(Role, f"player_{vision}")

```

```

1114         obs = all_obs[role]
1115
1116         mask = obs[-self.num_points :]
1117         obs = obs[:-self.num_points].reshape(3,-1)
1118         x = obs[:, -self.num_points:]
1119
1120         # Orientation
1121         z = np.zeros(2)
1122         o = 30 * np.array([0, 1])
1123         ax2.plot(o, z, z, color="r")
1124         ax2.plot(z, o, z, color="g")
1125         ax2.plot(z, z, o, color="b")
1126
1127
1128
1129         seen = x[:, mask == 0]
1130         unseen = x[:, mask == 1]
1131         ax2.scatter(seen[0], seen[1], seen[2], color="c")
1132         ax2.scatter(unseen[0], unseen[1], unseen[2], color="m")
1133         ax2.set_xlim([-100, 200])
1134         ax2.set_ylim([-100, 200])
1135         ax2.set_zlim([-100, 200])
1136
1137
1138         # Render Earth Frame
1139
1140         ax3 = fig.add_subplot(143)
1141
1142         N = len(self.orb.keys())
1143
1144         poss = np.stack([orb.r for orb in self.orb.values()]).to(self._OU_DIS).value
1145
1146         if "pos_hist" not in self.render_data.keys():
1147             self.render_data["pos_hist"] = []
1148
1149         self.render_data["pos_hist"].append(poss)
1150         pos_hist = np.array(self.render_data["pos_hist"])
1151
1152         for i in range(N):
1153             ax3.plot(pos_hist[:, i, 0], pos_hist[:, i, 1])
1154
1155
1156         ax3.set_title("Earth Centered Frame")
1157
1158
1159         # Render Hill Frame
1160
1161         ax4 = fig.add_subplot(144)
1162
1163         if "frames" not in self.render_data.keys():
1164             self.render_data["frames"] = []
1165
1166         if "chief_rel_poss" not in self.render_data.keys():
1167             self.render_data["chief_rel_poss"] = []
1168
1169         c_idx = list(self.orb.keys()).index("chief")
1170         frame = self.hills_frame(self.orb["chief"])
1171
1172         chief_rel_poss = (frame.T @ poss.T).T
1173         chief_rel_poss = chief_rel_poss - chief_rel_poss[c_idx, :]
1174
1175         self.render_data["chief_rel_poss"].append(chief_rel_poss)
1176         self.render_data["frames"].append(frame)
1177
1178         hills = np.array(self.render_data["chief_rel_poss"])
1179
1180         for i in range(N):
1181             ax4.plot(hills[:, i, 1], hills[:, i, 0])
1182
1183         chief = plt.Circle((0, 0), self._CHIEF_PERIMETER.to(self._OU_DIS).value, color='k', fill=False)
1184         ax4.add_patch(chief)
1185         ax4.set_title("Chief Centered Hill Frame")
1186
1187         # Draw
1188
1189         if render_mode == "rgb_array":
1190             canvas.draw()
1191
1192             data = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
1193             data = data.reshape(fig.canvas.get_width_height()[::-1] + (3,))
1194             plt.close(fig)

```

```

1195
1196         return data
1197
1198         plt.close(fig)
1199         return None
1200
1201     def close(self) -> None:
1202         pass
1203
1204     def state(self) -> None:
1205         raise NotImplementedError("State Not Implemented.")
1206
1207
1208 if __name__ == "__main__":
1209     from tqdm import tqdm
1210     from PIL import Image
1211
1212     env = MultInspect(
1213         num_deputies=4,
1214         _SIM_OUTER_PARAMETER=100<<u.m,
1215         _CHIEF_PERIMETER = 100 << u.m
1216     )
1217     env.seed(0)
1218     env.reset()
1219
1220     frames = []
1221     # for i in tqdm(range(env.max_episode_steps)):
1222     for i in tqdm(range(10)):
1223         act = {role: np.array([0,0,0,0,0,0]) for role in env.possible_agents}
1224         env.step(act)
1225         frames.append(env.render())
1226
1227     # Image.fromarray(frames[0])
1228
1229     imgs = [Image.fromarray(frame) for frame in frames]
1230     # duration is the number of milliseconds between frames; this is 40 frames per second
1231     model_dir = ""
1232     imgs[0].save(model_dir + f"_eval_run_{i}.gif", save_all=True, append_images=imgs[1:], duration=500,
1233                 loop=0)
1233     # %%

```

7 examples/MAInspection/Experiment

7.1 examples/MAInspection/Experiment/dash_app.yaml

```
1 interfaces:
2   Default: # Name of Interface
3     interface_class: DefaultInterface
4   models:
5     Default:
6       class_name: DefaultActor
7
8   Waypointer: # Name of Interface
9     interface_class: WaypointInterface
10
11   roles: [player_0] # If this is not supplied, assumed all roles
12   models:
13     PP0Waypointer:
14       class_name: SB_PP0WaypointActor
15       params:
16         policy_path: waypointer/MAInspect_20240213-193621/model.zip
17
18 directors:
19   PP0_Director:
20     class_name: SB3_PP0_Director
21     path: director/MAInspect_20240206-211327.zip
22     roles:
23       player_0:
24         classes: [PP0Waypointer]
```

7.2 examples/MAInspection/Experiment/dash_app_template.yaml

```
1 interfaces:
2   Default: # Name of Interface
3     interface_class: DefaultInterface
4     models:
5       Default:
6         class_name: DefaultActor
7
8   Waypointer: # Name of Interface
9     interface_class: WaypointInterface
10
11   roles: [player_0] # If this is not supplied, assumed all roles
12   models:
13     PPOWaypointer:
14       class_name: SB_PPOWaypointActor
15       params:
16         policy_path: <WAYPOINTER_PATH>
17
18 directors:
19   PPO_Director:
20     class_name: SB3_PPO_Director
21     path: <DIRECTOR_PATH>
22     roles:
23       player_0:
24         classes: [PPOWaypointer]
```

7.3 examples/MAInspection/Experiment/train_agents.yaml

```
1 waypointer:
2   env_params:
3     num_deputies: 1
4     _SB_SAFTY_MODE: true
5     _TRAIN_WAYPOINTER: true
6     _REWARD_FOR_LEAVING_PARAMETER: 0
7     _REWARD_FOR_LEAVING_OUTER_PERIMETER: 0
8     _CRASH_REWARD: 0
9     _REW_COV_MTRX: 22500 # 150^2
10    _PROX_RWD_SCALE: 10000
```

7.4 examples/MAInspection/Experiment/train_director.yaml

```
1  env_params:
2    num_deputies: 1
3    max_episode_steps: 150
4    _TRAIN_WAYPOINTER: false
5  COACH_params:
6    stochastic: true
7    FIXED_STEPS_PER_COM:
8      checkin_frequency: 20
9      allow_agent_break: False
10  ACTION_PADDING: 0
11  MIN_NEXT_ACTION_TIME: 19
12  MAX_NEXT_ACTION_TIME: 20
13  seed: 453413
14  Agents:
15    player_0:
16      class_name: "SB_PP0WaypointActor"
17      params:
18        policy_path:
```


8 examples/MAInspection/assets

8.1 examples/MAInspection/assets/typography.css

```
1 body {
2     font-family: sans-serif;
3 }
4
5 h1, h2, h3, h4, h5, h6 {
6     color: black
7 }
8
9 .resume {
10     display: none;
11 }
12
13 .button4 {
14     background-color: white; /* Green */
15     border: none;
16     color: black;
17     padding: 15px 32px;
18     text-align: left;
19     text-decoration: none;
20     display: inline-block;
21     font-size: 16px;
22     width: 100%;
23 }
24
25 .button4:hover {
26     background-color: #DDDDDD; /* Green */
27     color: white;
28 }
29
30
31 .float-container {
32     border: 3px solid #fff;
33     padding: 20px;
34     width: 600px;
35     margin: 0;
36     height: 300px;
37 }
38
39 .float-child {
40     width: 50%;
41     height: 100%;
42     float: left;
43     padding: 20px;
44     border: 2px solid red;
45 }
46
47 .agent_card {
48     float: left;
49     padding: 10px;
50     height: 150px;
51     width: 100%;
52     margin: auto;
53 }
54
55 .action_cards {
56     float: left;
57     padding: 10px;
58     width: 80%;
59     margin: auto;
60 }
61
62 .left_card {
63     float: left;
64     width: 25%;
65     height: 100%;
66     background-color: #aaa;
67     margin: auto;
68 }
69
70 .right_card {
71     float: left;
72     width: 70%;
73     height: 100%;
74     background-color: #bbb;
75     margin: auto;
```

```

76     }
77
78     .coa_card {
79         float: left;
80         width: 15%;
81         height: 100%;
82         background-color: #bbb;
83         margin:auto;
84     }
85
86     .current_plan tr {
87         background-color: #ccc;
88     }
89
90     .archived_plan tr {
91         background-color: #eee;
92     }
93
94     .archived_plan tr:hover {
95         background-color: #ddd;
96     }
97
98     /* Clear floats after the columns */
99     .agent_card_container:after {
100         content: "";
101         display: table;
102         clear: both;
103     }
104
105
106     /* Colors for sliders */
107
108     .rc-slider-handle-1 {
109         border-color: #01befe;
110     }
111
112     .rc-slider-handle-1.rc-slider-handle-click-focused:focus {
113         border-color: #01befe;
114     }
115
116     .rc-slider-handle-1:hover {
117         border-color: #01befe;
118     }
119
120     /* ===== */
121
122     .rc-slider-handle-2 {
123         border-color: #ffdd00;
124     }
125
126     .rc-slider-handle-2.rc-slider-handle-click-focused:focus {
127         border-color: #ffdd00;
128     }
129
130     .rc-slider-handle-2:hover {
131         border-color: #ffdd00;
132     }
133
134     /* ===== */
135
136     .rc-slider-handle-3 {
137         border-color: #ff7d00;
138     }
139
140     .rc-slider-handle-3.rc-slider-handle-click-focused:focus {
141         border-color: #ff7d00;
142     }
143
144     .rc-slider-handle-3:hover {
145         border-color: #ff7d00;
146     }
147
148     /* ===== */
149
150     .rc-slider-handle-4 {
151         border-color: #ff006d;
152     }
153
154     .rc-slider-handle-4.rc-slider-handle-click-focused:focus {
155         border-color: #ff006d;
156     }

```

```

157
158 .rc-slider-handle-4:hover {
159     border-color: #ff006d;
160 }
161
162 /* ===== */
163
164 .rc-slider-handle-5 {
165     border-color: #adff02;
166 }
167
168 .rc-slider-handle-5.rc-slider-handle-click-focused:focus {
169     border-color: #adff02;
170 }
171
172 .rc-slider-handle-5:hover {
173     border-color: #adff02;
174 }
175
176 /* ===== */
177
178 .rc-slider-handle-6 {
179     border-color: #8f00ff;
180 }
181
182 .rc-slider-handle-6.rc-slider-handle-click-focused:focus {
183     border-color: #8f00ff;
184 }
185
186 .rc-slider-handle-6:hover {
187     border-color: #8f00ff;
188 }
189
190 /* ===== */

```

9 examples/Multipolicy

9.1 examples/Multipolicy/Readme.md

```
1 # Multiploicy Director
2
3 In this example, we use StableBaselines3 and Supersuit to train two policies on the Waterworld
  environment from PettingZoo. One we train in a sparse food environment as an "explorer" agent, the
  other we train in a dense food and dense poison environment as a "dense" avoider agent. Finally, we
  train a director to select the policies each agent should use for the next 10 turns given their
  current observations.
4
5 In this example we see how to train policies on a PettingZoo environment, use those policies and that
  PettingZoo environment to establish a planning problem using the coach environment. We then set up
  a communications schedule to allow communication every 10 turns and train the director agent to
  select between the policies.
6
7 Finally, we construct a Dash app to display our solution.
8
9 This example uses all most entirely default functionality. For more advanced functionality see the
  MAInspection example.
10
11 
```

9.2 examples/Multipolicy/app.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #   http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import sys
16 import glob
17 import yaml
18 sys.path.insert(0, "../..")
19
20 from DASH.app import *
21
22 from pettingzoo.sisl import waterworld_v4
23 from utilities.PZWrapper import PettingZooEnv
24
25 # Logging
26 import logging
27 logging.basicConfig(level=logging.DEBUG, format="%(levelname)s: %(name)s: %(message)s")
28 logger = logging.getLogger(__name__)
29
30 pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
31                   name.startswith("pymunk")]
32 for log_handler in pymunk_loggers:
33     log_handler.setLevel(logging.INFO)
34
35 def get_env():
36     return PettingZooEnv(PZGame=waterworld_v4)
37
38 if __name__ == "__main__":
39     print("app - ##### RELOADING DASH APP #####")
40
41     MODEL_PATH = ""
42     DENSE_PATH = min(glob.iglob(os.path.join(MODEL_PATH, 'dense/*.zip')), key=os.path.getctime)
43     EXPLORE_PATH = min(glob.iglob(os.path.join(MODEL_PATH, 'explore/*.zip')), key=os.path.getctime)
44
45     with open(os.path.join("Experiment", "train_director.yaml"), "r") as f:
46         try:
47             params = yaml.safe_load(f)
48         except yaml.YAMLError as exc:
49             print(exc)
50
51     with open(os.path.join("Experiment", "dash_app.yaml"), "r") as f:
52         try:
53             params["actor_params"] = yaml.safe_load(f)
54         except yaml.YAMLError as exc:
55             print(exc)
56
57     # external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
58     env_factory = COACHIntegration(
59         env_creator=get_env,
60         COACHEnvClass=COACHEnvironment,
61         parameters=params
62     )
63
64     ct.env_factory = env_factory
65
66     locks = {
67         "action_card": False
68     }
69
70     ## App Layout
71     proxy_url = re.sub("{{port}}", "8050", os.environ["VSCODE_PROXY_URI"])
72     proxy_url = re.sub(os.environ["ACEHUB_BASEURL"], "", proxy_url)
73     app = Dash(serve_locally=True, requests_pathname_prefix=proxy_url)
74
75     app.layout = app_layout(env_factory)
76
77
```

78 `app.run(debug=True)`

9.3 examples/Multipolicy/train_agents.py

```
1  # Copyright (c) 2024 Mobius Logic, Inc.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #   http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from __future__ import annotations
16
17 from pettingzoo.sisl import waterworld_v4
18 import supersuit as ss
19 from stable_baselines3 import PPO
20 from stable_baselines3.ppo import MlpPolicy
21 from stable_baselines3.common.callbacks import EvalCallback
22 import os
23 import time
24 import glob
25 from tqdm import tqdm
26 import numpy as np
27 import yaml
28 import copy
29
30 import torch
31 CORES = 9
32 torch.set_num_threads(CORES)
33 torch.set_num_interop_threads(CORES)
34
35
36 # Code adapted from https://pettingzoo.farama.org/tutorials/sb3/waterworld/
37
38 ## We're going to train two policies, one that assumes a densely poisonous env
39 ## and one that requires sparse exploration.
40
41 def train_butterfly_supersuit(
42     env_fn,
43     steps: int = 10_000,
44     seed: int | None = 0,
45     num_vec_envs=1,
46     num_cpus=CORES,
47     learning_rate=1e-3,
48     batch_size=256,
49     model_dir="",
50     name="",
51     **env_kwargs
52 ):
53     model_path = os.path.join(model_dir, name)
54     os.makedirs(model_path, exist_ok=True)
55
56     # Train a single model to play as each agent in a cooperative Parallel environment
57     env = env_fn.parallel_env(**env_kwargs)
58     env.reset(seed=seed)
59
60     print(f"Starting training on {str(env.metadata['name'])}.")
61
62     env = ss.pettingzoo_env_to_vec_env_v1(env)
63     env = ss.concat_vec_envs_v1(env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
64                                base_class="stable_baselines3")
65
66     # Note: Waterworld's observation space is discrete (242,) so we use an MLP policy rather than CNN
67     model = PPO(
68         MlpPolicy,
69         env,
70         verbose=3,
71         learning_rate=learning_rate,
72         batch_size=batch_size,
73         tensorboard_log=os.path.join("./tensorboard_log/", name)
74     )
75
76     eval_env = env_fn.parallel_env(**env_kwargs)
77     eval_env.reset(seed=seed)
78     eval_env = ss.pettingzoo_env_to_vec_env_v1(eval_env)
```

```

78     eval_env = ss.concat_vec_envs_v1(eval_env, num_vec_envs=1, num_cpus=num_cpus,
79                                     base_class="stable_baselines3")
80     eval_callback = EvalCallback(eval_env, verbose=1, eval_freq=10000)
81
82     weight_name = f"{env.unwrapped.metadata.get('name')}-{time.strftime('%Y%m%d-%H%M%S')}"
83     weight_path = os.path.join(model_path, weight_name)
84
85     model_args = {
86         "name": name,
87         "load_class": "SB_PPO PoliciesActor",
88         "env_params": env_kwargs
89     }
90
91     with open(weight_path + ".yaml", "w") as f:
92         yaml.dump(model_args, f, default_flow_style=False)
93
94     model.learn(total_timesteps=steps, callback=eval_callback)
95     model.save(weight_path)
96
97     print("Model has been saved.")
98     print(f"Finished training on {str(env.unwrapped.metadata['name'])}.")
99
100     env.close()
101
102     return weight_path
103
104
105 def eval(
106     env_fn,
107     model_path,
108     num_games: int = 10,
109     render_mode: str | None = None,
110     **env_kwargs
111 ):
112     # Evaluate a trained agent vs a random agent
113     env = env_fn.env(render_mode=render_mode, **env_kwargs)
114
115     print(
116         f"\nStarting evaluation on {str(env.metadata['name'])} (num_games={num_games},
117             render_mode={render_mode})"
118     )
119
120     model = PPO.load(model_path)
121
122     cum_rewards = []
123
124     # Note: We train using the Parallel API but evaluate using the AEC API
125     # SB3 models are designed for single-agent settings, we get around this by using the same model for
126     # every agent
127     for i in tqdm(range(num_games)):
128         env.reset(seed=i)
129         rewards = {agent: 0 for agent in env.possible_agents}
130
131         for agent in env.agent_iter():
132             obs, reward, termination, truncation, info = env.last()
133
134             for a in env.agents:
135                 rewards[a] += env.rewards[a]
136             if termination or truncation:
137                 for a in env.agents:
138                     cum_rewards.append(rewards[a])
139                 break
140             else:
141                 act = model.predict(obs, deterministic=True)[0]
142                 env.step(act)
143         env.close()
144
145     avg_reward = np.mean(cum_rewards)
146     std_reward = np.std(cum_rewards)
147
148     with open(model_path + ".txt", "a") as f:
149         f.writelines(f"\n\n{env_kwargs}\n")
150         f.writelines(f"\t Avg reward: {avg_reward}, std: {std_reward}\n")
151         f.writelines(f"\t Rewards: {cum_rewards}\n")
152
153     print(f"\t Avg reward: {avg_reward}, std: {std_reward}")
154     return avg_reward
155

```



```

156
157 if __name__ == "__main__":
158     env_fn = waterworld_v4
159     env_kwargs = {}
160
161     STEPS = 1000
162
163     ## Train Dense Eater
164     ## Note: We want one eater here so that it doesn't learn to just sit there and hope for food.
165     model_path_dense = train_butterfly_supersuit(
166         env_fn=env_fn,
167         steps=STEPS,
168         seed=0,
169         n_pursuers=1,
170         n_evaders=10,
171         n_poisons=40,
172         model_dir="",
173         name="dense"
174     )
175
176     ## Train Explore Eater
177     model_path_explore = train_butterfly_supersuit(
178         env_fn=env_fn,
179         steps=STEPS,
180         seed=0,
181         n_pursuers=3,
182         n_evaders=5,
183         n_poisons=5,
184         model_dir="",
185         name="explore")
186
187     for model_path in [model_path_dense, model_path_explore]:#, model_path_explore]:
188         eval(env_fn=env_fn,
189             model_path=model_path,
190             n_pursuers=3,
191             n_evaders=20,
192             n_poisons=30,
193             )
194
195         eval(env_fn=env_fn,
196             model_path=model_path,
197             n_pursuers=3,
198             n_evaders=5,
199             n_poisons=5,
200             )
201
202         eval(env_fn=env_fn,
203             model_path=model_path,
204             n_pursuers=3,
205             n_evaders=15,
206             n_poisons=15,
207             )

```

9.4 examples/Multipolicy/train_director.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 from __future__ import annotations
15 import sys
16 sys.path.insert(0, "../..")
17
18 import os
19 import time
20 import glob
21 from tqdm import tqdm
22 import numpy as np
23 from PIL import Image
24 import yaml
25
26 from utilities.PZWrapper import PettingZooEnv
27 from coach import COACHEnvironment
28 from env import COACH_PettingZoo
29
30 from stable_baselines3 import PPO
31 from stable_baselines3.ppo import MlpPolicy
32 from stable_baselines3.common.callbacks import EvalCallback
33 import supersuit as ss
34
35 from pettingzoo.sisl import waterworld_v4
36
37 import torch
38 CORES = 6
39 torch.set_num_threads(CORES)
40 torch.set_num_interop_threads(CORES)
41
42 import logging
43 logging.basicConfig()
44 logging.getLogger().setLevel(logging.INFO)
45 logger = logging.getLogger(__name__)
46
47 pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
48                   name.startswith("pymunk")]
49 for log_handler in pymunk_loggers:
50     log_handler.setLevel(logging.INFO)
51
52 # SB code adapted from https://pettingzoo.farama.org/tutorials/sb3/waterworld/
53
54 ## We're going to train two policies, one that assumes a densely poisonous env
55 ## and one that requires sparse exploration.
56 # %%
57 def train(
58     env_fn,
59     steps: int = 10_000,
60     seed: int | None = 0,
61     num_vec_envs=1,
62     num_cpus=1,
63     learning_rate=1e-3,
64     batch_size=256,
65     model_dir=""
66 ):
67     os.makedirs(model_dir, exist_ok=True)
68
69     # Train a single model to play as each agent in a cooperative Parallel environment
70     env = env_fn()
71     env.reset(seed=seed)
72     print(env.possible_agents)
73
74     logger.info(f"Starting training on {str(env.metadata['name'])}.")
75
76     env = ss.pettingzoo_env_to_vec_env_v1(env)
77     env = ss.concat_vec_envs_v1(env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
78                                base_class="stable_baselines3")
```

```

77
78 # Note: Waterworld's observation space is discrete (242,) so we use an MLP policy rather than CNN
79 model = PP0(
80     MlpPolicy,
81     env,
82     verbose=3,
83     learning_rate=learning_rate,
84     batch_size=batch_size,
85     tensorboard_log=os.path.join("./tensorboard_log/", model_dir.split("/")[-1])
86 )
87
88 eval_env = env_fn()
89 eval_env = ss.pettingzoo_env_to_vec_env_v1(eval_env)
90 eval_env = ss.concat_vec_envs_v1(eval_env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
91     base_class="stable_baselines3")
92 eval_callback = EvalCallback(eval_env, verbose=1, eval_freq=500)
93
94 model.learn(total_timesteps=steps, callback=eval_callback)
95
96 model_name = f"{env.unwrapped.metadata.get('name')}_{'time.strftime('%Y%m%d-%H%M%S')}"
97 model_path = os.path.join(model_dir, model_name)
98 model.save(model_path)
99
100 logger.info("Model has been saved.")
101 logger.info(f"Finished training on {str(env.unwrapped.metadata['name'])}.")
102
103 env.close()
104
105 return model_path + ".zip"
106
107 def eval(
108     env_fn,
109     model_path,
110     num_games: int = 10,
111     render = False,
112     **env_kwargs
113 ):
114     # Evaluate a trained agent vs a random agent
115     env = env_fn(**env_kwargs)
116
117     logger.info(
118         f"\nStarting evaluation on {str(env.metadata['name'])} (num_games={num_games}, render={render})"
119     )
120
121     model = PP0.load(model_path)
122
123     cum_rewards = []
124
125     # Note: We train using the Parallel API but evaluate using the AEC API
126     # SB3 models are designed for single-agent settings, we get around this by using the same model for every agent
127     for i in tqdm(range(num_games)):
128         if render:
129             env.coa_env.start_rendering()
130
131         obs, info = env.reset(seed=i)
132
133         rewards = {agent: 0 for agent in env.possible_agents}
134         running = True
135         j = 0
136         while running:
137             j += 1
138             act = {"director": model.predict(obs["director"], deterministic=False)[0]}
139
140             logger.debug("Obs From Director: %s", obs["director"]) # DEBUG
141             logger.debug("Action From Director: %s", act) # DEBUG
142
143             obs, reward, term, trunc, info = env.step(act)
144             for a in env.agents:
145                 rewards[a] += reward[a]
146
147             if all([a or b for a,b in zip(term.values(), trunc.values())]):
148                 for a in env.agents:
149                     cum_rewards.append(rewards[a])
150
151                 running = False
152
153         if render:
154             frames = env.coa_env.state.trajectory.frames
155             imgs = [Image.fromarray(frame) for frame in frames]
156             # duration is the number of milliseconds between frames; this is 40 frames per second

```

```

156         model_dir = model_path.split(".")[0]
157         imgs[0].save(model_dir + f"_eval_run_{i}.gif", save_all=True, append_images=imgs[1:],
            duration=500, loop=0)
158
159     env.close()
160
161     avg_reward = np.mean(cum_rewards)
162     std_reward = np.std(cum_rewards)
163
164     with open(model_path + ".txt", "a") as f:
165         f.writelines(f"\n\n{env_kwargs}\n")
166         f.writelines(f"\t Avg reward: {avg_reward}, std: {std_reward}\n")
167         f.writelines(f"\t Rewards: {cum_rewards}\n")
168
169     logger.info(f"\t Avg reward: {avg_reward}, std: {std_reward}")
170     return avg_reward
171
172
173 def get_env():
174     return PettingZooEnv(PZGame=waterworld_v4)
175
176 if __name__ == "__main__":
177     MODEL_PATH = ""
178     DENSE_PATH = min(glob.iglob(os.path.join(MODEL_PATH, 'dense/*.zip')), key=os.path.getctime)
179     EXPLORE_PATH = min(glob.iglob(os.path.join(MODEL_PATH, 'explore/*.zip')), key=os.path.getctime)
180
181     with open(os.path.join("Experiment", "train_director.yaml"), "r") as stream:
182         try:
183             params = yaml.safe_load(stream)
184         except yaml.YAMLError as exc:
185             print(exc)
186
187     for agent, param in params["COACH_params"]["Agents"].items():
188         param["params"]["policy_paths"]["dense"] = DENSE_PATH
189         param["params"]["policy_paths"]["explore"] = EXPLORE_PATH
190
191     env = COACH_PettingZoo(env_creator=get_env, COACHEnvClass=COACHEnvironment)
192
193     def get_env_pz():
194         env = COACH_PettingZoo(env_creator=get_env, COACHEnvClass=COACHEnvironment)
195         env.augment(params)
196         env.reset()
197         return env
198
199     model_path = train(
200         get_env_pz,
201         steps=100,
202         seed=0,
203         model_dir="director"
204     )
205
206     model_path = min(glob.iglob(os.path.join(MODEL_PATH, 'director/*.zip')), key=os.path.getctime)
207     rew = eval(
208         get_env_pz,
209         model_path=model_path,
210         num_games=1,
211         render=True
212     )
213
214     with open(os.path.join("Experiment", "dash_app_template.yaml"), 'r') as f:
215         with open(os.path.join("Experiment", "dash_app.yaml"), 'w') as g:
216             for line in f.readlines():
217                 line = line.replace("<DENSE_PATH>", DENSE_PATH)
218                 line = line.replace("<EXPLORE_PATH>", EXPLORE_PATH)
219                 line = line.replace("<DIRECTOR_PATH>", model_path)
220                 g.write(line)
221
222 # %%

```

10 examples/Multipolicy/Experiment

10.1 examples/Multipolicy/Experiment/dash_app.yaml

```
1 interfaces:
2   Default: # Name of Interface
3     interface_class: DefaultInterface
4   models:
5     Default:
6       class_name: DefaultActor
7
8   Random: # Name of Interface
9     interface_class: RandomActionInterface
10  models:
11    Random:
12      class_name: RandomActor
13
14  Dense_v_Sparse: # Name of Interface
15    interface_class: SBPolicyInterface
16    interface_parameters:
17      n_policies: 2
18      max_action_len: 10
19
20  roles: [pursuer_0, pursuer_1, pursuer_2] # If this is not supplied, assumed all roles
21  models:
22    Dense_v_Sparse_v1:
23      class_name: SB_PP0PoliciesActor
24      params:
25        policy_paths:
26          dense: dense/waterworld_v4_20240213-185458.zip
27          explore: explore/waterworld_v4_20240213-185504.zip
28        max_action_len: 10
29
30    Dense_v_Sparse_v2:
31      class_name: SB_PP0PoliciesActor
32      params:
33        policy_paths:
34          dense: dense/waterworld_v4_20240213-185458.zip
35          explore: explore/waterworld_v4_20240213-185504.zip
36        max_action_len: 10
37
38  Tall_v_Short:
39    interface_class: SBPolicyInterface
40    interface_parameters:
41      n_policies: 2
42      max_action_len: 10
43
44  models:
45    Tall_v_Short:
46      class_name: SB_PP0PoliciesActor
47      params:
48        policy_paths:
49          dense: dense/waterworld_v4_20240213-185458.zip
50          explore: explore/waterworld_v4_20240213-185504.zip
51        max_action_len: 10
52  roles: [pursuer_0, pursuer_1, pursuer_2] # If this is not supplied, assumed all roles
53
54  directors:
55    PP0_Director:
56      class_name: SB3_PP0_Director
57      path: director/waterworld_v4_20240213-190312.zip
58      roles:
59        pursuer_0:
60          classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
61        pursuer_1:
62          classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
63        pursuer_2:
64          classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
```

10.2 examples/Multipolicy/Experiment/dash_app_template.yaml

```
1 interfaces:
2   Default: # Name of Interface
3     interface_class: DefaultInterface
4     models:
5       Default:
6         class_name: DefaultActor
7
8   Random: # Name of Interface
9     interface_class: RandomActionInterface
10    models:
11      Random:
12        class_name: RandomActor
13
14   Dense_v_Sparse: # Name of Interface
15     interface_class: SBPolicyInterface
16     interface_parameters:
17       n_policies: 2
18       max_action_len: 10
19
20   roles: [pursuer_0, pursuer_1, pursuer_2] # If this is not supplied, assumed all roles
21   models:
22     Dense_v_Sparse_v1:
23       class_name: SB_PP0PoliciesActor
24       params:
25         policy_paths:
26           dense: <DENSE_PATH>
27           explore: <EXPLORE_PATH>
28         max_action_len: 10
29
30     Dense_v_Sparse_v2:
31       class_name: SB_PP0PoliciesActor
32       params:
33         policy_paths:
34           dense: <DENSE_PATH>
35           explore: <EXPLORE_PATH>
36         max_action_len: 10
37
38   Tall_v_Short:
39     interface_class: SBPolicyInterface
40     interface_parameters:
41       n_policies: 2
42       max_action_len: 10
43
44   models:
45     Tall_v_Short:
46       class_name: SB_PP0PoliciesActor
47       params:
48         policy_paths:
49           dense: <DENSE_PATH>
50           explore: <EXPLORE_PATH>
51         max_action_len: 10
52   roles: [pursuer_0, pursuer_1, pursuer_2] # If this is not supplied, assumed all roles
53
54 directors:
55   PP0_Director:
56     class_name: SB3_PP0_Director
57     path: <DIRECTOR_PATH>
58     roles:
59       pursuer_0:
60         classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
61       pursuer_1:
62         classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
63       pursuer_2:
64         classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
```

10.3 examples/Multipolicy/Experiment/train_agents.yaml

```
1 train_agents:
2   dense:
3     env_params:
4       n_evaders: 10
5       n_poisons: 40
6       n_pursuers: 1
7   explore:
8     env_params:
9       n_evaders: 5
10      n_poisons: 5
11      n_pursuers: 3
```

10.4 examples/Multipolicy/Experiment/train_director.yaml

```
1  env_params:
2    n_pursuers: 3
3    max_cycles: 30
4  COACH_params:
5    stochastic: true
6    FIXED_STEPS_PER_COM:
7      checkin_frequency: 10
8      allow_agent_break: False
9    ACTION_PADDING: 0
10   MIN_NEXT_ACTION_TIME: 1
11   MAX_NEXT_ACTION_TIME: 10
12   seed: 453413
13  Agents:
14    pursuer_0:
15      class_name: "SB_PPOPoliciesActor"
16      params:
17        max_action_len: 10
18        policy_paths:
19          dense:
20            explore:
21    pursuer_1:
22      class_name: "SB_PPOPoliciesActor"
23      params:
24        max_action_len: 10
25        policy_paths:
26          dense:
27            explore:
28    pursuer_2:
29      class_name: "SB_PPOPoliciesActor"
30      params:
31        max_action_len: 10
32        policy_paths:
33          dense:
34            explore:
```


11 examples/Multipolicy/assets

11.1 examples/Multipolicy/assets/typography.css

```
1  body {
2      font-family: sans-serif;
3  }
4
5  h1, h2, h3, h4, h5, h6 {
6      color: black
7  }
8
9  .resume {
10     display: none;
11 }
12
13 .button4 {
14     background-color: white; /* Green */
15     border: none;
16     color: black;
17     padding: 15px 32px;
18     text-align: left;
19     text-decoration: none;
20     display: inline-block;
21     font-size: 16px;
22     width: 100%;
23 }
24
25 .button4:hover {
26     background-color: #DDDDDD; /* Green */
27     color: white;
28 }
29
30
31 .float-container {
32     border: 3px solid #fff;
33     padding: 20px;
34     width: 600px;
35     margin: 0;
36     height: 300px;
37 }
38
39 .float-child {
40     width: 50%;
41     height: 100%;
42     float: left;
43     padding: 20px;
44     border: 2px solid red;
45 }
46
47 .agent_card {
48     float: left;
49     padding: 10px;
50     height: 150px;
51     width: 100%;
52     margin: auto;
53 }
54
55 .action_cards {
56     float: left;
57     padding: 10px;
58     width: 80%;
59     margin: auto;
60 }
61
62 .left_card {
63     float: left;
64     width: 25%;
65     height: 100%;
66     background-color: #aaa;
67     margin: auto;
68 }
69
70 .right_card {
71     float: left;
72     width: 70%;
73     height: 100%;
74     background-color: #bbb;
75     margin: auto;
```

```

76     }
77
78     .coa_card {
79         float: left;
80         width: 15%;
81         height: 100%;
82         background-color: #bbb;
83         margin:auto;
84     }
85
86     .current_plan tr {
87         background-color: #ccc;
88     }
89
90     .archived_plan tr {
91         background-color: #eee;
92     }
93
94     .archived_plan tr:hover {
95         background-color: #ddd;
96     }
97
98     /* Clear floats after the columns */
99     .agent_card_container:after {
100         content: "";
101         display: table;
102         clear: both;
103     }
104
105
106     /* Colors for sliders */
107
108     .rc-slider-handle-1 {
109         border-color: #01befe;
110     }
111
112     .rc-slider-handle-1.rc-slider-handle-click-focused:focus {
113         border-color: #01befe;
114     }
115
116     .rc-slider-handle-1:hover {
117         border-color: #01befe;
118     }
119
120     /* ===== */
121
122     .rc-slider-handle-2 {
123         border-color: #ffdd00;
124     }
125
126     .rc-slider-handle-2.rc-slider-handle-click-focused:focus {
127         border-color: #ffdd00;
128     }
129
130     .rc-slider-handle-2:hover {
131         border-color: #ffdd00;
132     }
133
134     /* ===== */
135
136     .rc-slider-handle-3 {
137         border-color: #ff7d00;
138     }
139
140     .rc-slider-handle-3.rc-slider-handle-click-focused:focus {
141         border-color: #ff7d00;
142     }
143
144     .rc-slider-handle-3:hover {
145         border-color: #ff7d00;
146     }
147
148     /* ===== */
149
150     .rc-slider-handle-4 {
151         border-color: #ff006d;
152     }
153
154     .rc-slider-handle-4.rc-slider-handle-click-focused:focus {
155         border-color: #ff006d;
156     }

```

```

157
158 .rc-slider-handle-4:hover {
159     border-color: #ff006d;
160 }
161
162 /* ===== */
163
164 .rc-slider-handle-5 {
165     border-color: #adff02;
166 }
167
168 .rc-slider-handle-5.rc-slider-handle-click-focused:focus {
169     border-color: #adff02;
170 }
171
172 .rc-slider-handle-5:hover {
173     border-color: #adff02;
174 }
175
176 /* ===== */
177
178 .rc-slider-handle-6 {
179     border-color: #8f00ff;
180 }
181
182 .rc-slider-handle-6.rc-slider-handle-click-focused:focus {
183     border-color: #8f00ff;
184 }
185
186 .rc-slider-handle-6:hover {
187     border-color: #8f00ff;
188 }
189
190 /* ===== */

```

12 utilities

12.1 utilities/PZParams.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15
16 """PettingZoo Parameter Class """
17
18 import argparse
19 import logging
20 from argparse import Namespace # for type hinting
21 from collections.abc import Callable # for type hinting
22 from typing import Any, Optional # for type hinting
23
24 class EnvParams:
25     """Base class for environment params."""
26
27     def __init__(self, args: Any, param_section_name: str = "env_params") -> None:
28         self.args = args
29         self._params = getattr(args, param_section_name, {})
30
31     def __getitem__(self, key: str) -> Any:
32         """Return value stored for key from the params dict."""
33         return self._params[key]
34
35     def __setitem__(self, key: str, value: Any) -> None:
36         """Set the value for key in the params dict."""
37         self._params[key] = value
38
39     def get(self, key: str, default: Optional[Any] = None) -> Any:
40         """Return value for key from the params dict, or None if it doesn't exist."""
41         try:
42             return self._params[key]
43         except KeyError:
44             return default
45
46     def get_mutated_params(self) -> "EnvParams":
47         """Return a mutated copy of the params"""
48         raise NotImplementedError(
49             f"get_mutated_params has not been implemented in {type(self)}"
50         )
51
52     def checkpoint(self, folder: str) -> None:
53         """Save a checkpoint in the given folder."""
54         raise NotImplementedError(
55             f"checkpoint has not been implemented in {type(self)}"
56         )
57
58     def reload(self, folder: str) -> None:
59         """Read a checkpoint from the given folder."""
60         raise NotImplementedError(f"reload has not been implemented in {type(self)}")
61
62
63 #####
64 ## Auxiliary Functions
65 #####
66 def get_env_param_class(args: Namespace) -> Callable[..., Any]:
67     """Returns the class to use, based on input arguments
68
69     Parameters
70     -----
71     args: argparse.Namespace
72         arguments that were passed to the `main()` function
73
74     Returns
75     -----
```

```

76     class
77         the class to use in creating env parameter objects
78         """
79     return PettingZooEnvParams
80
81
82 #####
83 ## Main Class
84 #####
85 class PettingZooEnvParams(EnvParams):
86     """Parameters for PettingZoo Environments"""

```

12.2 utilities/PZWrapper.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 # The following code is modified from Farama-Foundation/PettingZoo
16 # (https://github.com/Farama-Foundation/PettingZoo)
17 # under the MIT License.
18
19 """PettingZoo Wrapper"""
20
21 import argparse # for type hinting
22 import logging
23 from collections.abc import Callable # for type hinting
24 from typing import Any, Union, cast # for type hinting
25
26 from gymnasium.wrappers import FlattenObservation
27 from pettingzoo.classic import connect_four_v3 as PZGame # type: ignore[import]
28 from pettingzoo.sisl import waterworld_v4 as PZGame
29
30 logging.getLogger("pettingzoo.utils.env_logger").setLevel(logging.WARNING)
31 logger = logging.getLogger(__name__)
32
33 import copy
34
35 import numpy as np
36 from gymnasium.spaces import Box, Discrete, flatten, flatten_space, unflatten
37 from numpy.typing import NDArray
38 from pettingzoo.utils.conversions import aec_to_parallel
39 from pettingzoo.utils.env import AECEnv, ParallelEnv
40
41 from .PZParams import PettingZooEnvParams
42
43
44 #####
45 ## Auxiliary Functions
46 #####
47 def softmax(x):
48     exp = np.exp(x)
49     return exp / cast(float, np.exp(x).sum())
50
51
52 @unflatten.register(Discrete)
53 def _unflatten_discrete(
54     space: Discrete, x: Union[NDArray[np.int64], NDArray[float]]
55 ) -> np.int64:
56     nonzero = np.nonzero(x)
57     if len(nonzero[0]) == 0:
58         raise ValueError(
59             f"{x} is not a valid one-hot encoded vector and can not be unflattened to space {space}. "
60             "Not all valid samples in a flattened space can be unflattened."
61         )
62
63     act = np.argmax(x)
64
65     return space.start + act
66
67
68 #####
69 ## Factories
70 #####
71 def get_env_class(args: argparse.Namespace) -> Callable[..., Any]:
72     """Returns the class to use, based on input arguments
73
74     Parameters
75     -----
76     args: argparse.Namespace
77         arguments that were passed to the `main()` function
78
79 """
```

```

79     Returns
80     -----
81     class
82         the class to use in creating env objects
83     """
84     return PettingZooEnv
85
86
87 ## This is what you want to edit if you add wrappers, or change
88 ## things about the env parameters.
89 def env_creator(**kwargs):
90     return PZGame.env(**kwargs)
91
92
93 #####
94 ## Main Class
95 #####
96 class PettingZooEnv:
97     def __init__(self, PZGame) -> None:
98         """ """
99         def env_creator(**kwargs):
100             return PZGame.parallel_env(**kwargs)
101
102         self.env_creator = env_creator
103         # If you have wrappers define a function to set it up properly
104         self.standard_params = {"render_mode": "rgb_array"}
105         self.current_params = None
106
107     def __getattr__(self, name):
108         """Returns an attribute with ``name``, unless ``name`` starts with an underscore."""
109         if name == "env":
110             if "env" not in self.__dict__.keys():
111                 self.__dict__["env"] = None
112                 return self.__dict__["env"]
113
114         return getattr(self.env, name)
115
116     def unflatten_or_none(self, action_space, action):
117         if action is not None:
118             return unflatten(action_space, action)
119         else:
120             return None
121
122     def step(self, act):
123         act = {agt: self.unflatten_or_none(self.env.action_space(agt), v) for agt, v in act.items()}
124
125         # logger.info(f"action: {str(act)}")
126         if self.parallel:
127             return self.env.step(act)
128         else:
129             return self.env.step(act[self.env.agent_selection])
130
131         obs = {
132             agt: flatten(self.env.observation_space(agt), self.env.observe(agt))
133             for agt in self.env.possible_agents
134         }
135
136         # logger.info(f"action: {str(obs)}, {self.env.terminations}, {self.env.truncations}")
137         return (
138             obs,
139             self.env.rewards,
140             self.env.terminations,
141             self.env.truncations,
142             self.env.infos,
143         )
144
145     def reset(self):
146         if self.parallel:
147             return self.env.reset()
148
149         self.env.reset()
150         # print(obs)
151         obs = {
152             agt: self.env.observation_space(agt)
153             for agt in self.env.possible_agents
154         }
155         # return obs, info
156         return obs, self.env.info
157
158     def augment(self, params):
159         self.current_params = copy.copy(self.standard_params)

```

```

160
161     for k, v in params.items():
162         self.current_params[k] = v
163
164     self.env = self.env_creator(**self.current_params)
165     if isinstance(self.env, ParallelEnv):
166         self.parallel = True
167     else:
168         self.parallel = False
169
170     self.env.reset()
171
172     self.action_spaces = dict()
173     self.observation_spaces = dict()
174
175     self.action_type = dict()
176     for agt in self.env.possible_agents:
177         self.action_spaces[agt] = flatten_space(self.env.action_space(agt))
178         self.observation_spaces[agt] = flatten_space(
179             self.env.observation_space(agt)
180         )
181         self.action_type[agt] = type(self.env.action_space(agt))
182
183     def seed(self, seed):
184         if hasattr(self.env, "seed"):
185             self.env.seed(seed)
186
187     def render(self, *args, **kwargs):
188         if kwargs.get("close", False):
189             self.env.close()
190             return
191
192         return self.env.render()

```


12.3 utilities/iotools.py

```
1 # Copyright (c) 2023 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15
16 """Contains various utility functions/classes used in the project"""
17 import csv
18 import json
19 import os
20 from collections.abc import Callable # for type hinting
21 from typing import Any, Union, cast, NewType # for type hinting
22
23 import numpy as np
24
25 EnvId = NewType("EnvId", str)
26 PathString = str
27
28
29 #####
30 ## Auxiliary Functions
31 #####
32 ### Turn dictionary with keyed tuple into csv
33 def save_keyed_tuple(
34     dct: dict[tuple[EnvId, EnvId], float],
35     filename: PathString,
36     do_sort: bool = True,
37 ) -> None:
38     """
39     Writes a csv file of a dict that is indexed by a tuple (a matrix)
40
41     Parameters
42     -----
43     dct : dict
44         Dict of values with keys given as tuples
45     filename : str
46         Name of file to save to
47     do_sort : boolean, optional
48         whether to sort the output keys
49         This is assuming they are of the form "Env_#"
50
51     Side-Effects
52     -----
53     None
54
55     Returns
56     -----
57     None
58
59     Notes
60     -----
61     As an example, for a dict of:
62     dct = {(a,a): val_aa, (a,b): val_ab,
63           (b,a): val_ba, (b,b): val_bb,
64           (c,a): val_ca, (c,b): val_cb,
65           (d,a): val_da, (d,b): val_db}
66     The output is:
67     ,a,b
68     a,val_aa,val_ab
69     b,val_ba,val_bb
70     c,val_ca,val_cb
71     d,val_da,val_db
72     Note that the ',' at the start of the first line is intentional
73     to provide an empty cell in the csv format so the data forms a
74     rectangular matrix.
75     """
76     # split (l, r) keys into lists of l and r
77     l_keys = list({first for first, second in dct})
78     r_keys = list({second for first, second in dct})
```

```

79
80 # sort by the numeric part of Env_X
81 if do_sort:
82     l_keys.sort(key=lambda x: int(x.split("_")[1]))
83     r_keys.sort(key=lambda x: int(x.split("_")[1]))
84
85 with open(filename, mode="w", newline="", encoding="utf8") as file:
86     csvfile = csv.writer(
87         file, delimiter=";", quoting=csv.QUOTE_MINIMAL, lineterminator="\n"
88     )
89     header = [""] + r_keys # [""] to add blank entry to csv row
90     csvfile.writerow(header)
91
92     for first in l_keys:
93         row = [first] + [dct.get((first, second), "") for second in r_keys]
94         csvfile.writerow(row)
95
96
97 def load_keyed_tuple(
98     filename: PathString,
99     format_function: Callable[[str], Any] = lambda x: x,
100 ) -> dict[tuple[EnvId, EnvId], Any]:
101     """
102     Read a tuple-indexed dict (a matrix) from a csv file.
103
104     This is the reverse of save_keyed_tuple
105     The format_function is used to convert the values from string to
106     whatever format is desirable. The default leaves it as a string.
107     A typical choice would be float.
108
109     Parameters
110     -----
111     filename : str
112         Name of file to save to
113     format_function: callable, optional
114         This is applied to values (but not keys) read from the file
115
116     Returns
117     -----
118     dict
119         The dict that was read
120
121     Notes
122     -----
123     For a file with the following:
124     ,a,b
125     a,val_aa,val_ab
126     b,val_ba,val_bb
127     c,val_ca,val_cb
128     d,val_da,val_db
129     The output is:
130     dct = {(a,a): val_aa, (a,b): val_ab,
131            (b,a): val_ba, (b,b): val_bb,
132            (c,a): val_ca, (c,b): val_cb,
133            (d,a): val_da, (d,b): val_db}
134     """
135     dct: dict[tuple[EnvId, EnvId], Any] = {}
136     with open(filename, mode="r", newline="", encoding="utf8") as file:
137         csvfile = csv.reader(file, delimiter=";")
138         # first line has an empty spot to account for alignment
139         # of columns. So, we ignore that
140         r_keys = next(csvfile)[1:]
141         for l_key, *values in csvfile:
142             for r_key, value in zip(r_keys, values):
143                 # appease type checker
144                 l_key = cast(EnvId, l_key)
145                 r_key = cast(EnvId, r_key)
146                 dct[(l_key, r_key)] = format_function(value)
147     return dct
148
149
150 #####
151 ## Numpy Encoders for JSON
152 #####
153 ## Recursively encodes objects with a reprJSON function
154 # https://stackoverflow.com/questions/5160077/encoding-nested-python-object-in-json
155 #
156 # Encdoing numpy objects:
157 # https://stackoverflow.com/questions/26646362/numpy-array-is-not-json-serializable
158 #
159 # Decoding objects

```

```

160 # https://stackoverflow.com/questions/48991911/how-to-write-a-custom-json-decoder-for-a-complex-object
161
162 # Usage
163 # with open(filename, 'w') as jsonfile:
164 #     json.dump(edge, jsonfile, cls=NumpyEncoder)
165 # with open(filename, 'r') as jsonfile:
166 #     edge1 = json.load(jsonfile, cls=NumpyDecoder)
167
168
169 class NumpyEncoder(json.JSONEncoder):
170     """Encode numpy data for JSON writer"""
171
172     def default(self, o): # type: ignore # this is called by the JSON library
173         if isinstance(o, np.integer):
174             return {"np.integer": int(o)}
175         if isinstance(o, np.floating):
176             return {"np.floating": float(o)}
177         if isinstance(o, np.ndarray):
178             return {"np.array": o.tolist()}
179         return json.JSONEncoder.default(self, o)
180
181
182 class NumpyDecoder(json.JSONDecoder):
183     """Decode numpy data from JSON"""
184
185     def __init__(self, *args, **kwargs): # type: ignore # this is called by the JSON library
186         json.JSONDecoder.__init__(self, object_hook=self.numpy_hook, *args, **kwargs)
187
188     def numpy_hook(self, dct): # type: ignore # this is called by the JSON library
189         """Convert dict with numpy data to numpy object"""
190         if "np.integer" in dct:
191             return np.int_(dct["np.integer"])
192         if "np.floating" in dct:
193             return np.float_(dct["np.floating"])
194         if "np.array" in dct:
195             return np.array(dct["np.array"])
196         return dct
197
198
199 #####
200 ## TensorFlow Logging
201 #####
202 class TBWriter:
203     def __init__(self, args: dict[str, Any]) -> None:
204         """ """
205         import time
206
207         from tensorflow import summary # type: ignore[import]
208
209         self.summary = summary
210         log_dir = args["log_dir"]
211
212         now = time.localtime()
213         subdir = time.strftime("%d-%b-%Y-%H.%M.%S", now)
214
215         self.summary_dir = os.path.join(log_dir, subdir)
216         self.summary_writer: dict[str, summary.SummaryWriter] = {}
217
218     def create_scalar_writer(self, name: str) -> None:
219         new_dir = os.path.join(self.summary_dir, name)
220         self.summary_writer[name] = self.summary.create_file_writer(new_dir)
221
222     def write_item(self, name: str, label: str, data: Any, step: int) -> None:
223         if label not in self.summary_writer.keys():
224             self.create_scalar_writer(label)
225
226         with self.summary_writer[label].as_default():
227             if isinstance(data, (np.ndarray, np.generic)):
228                 data = data.item()
229
230             if hasattr(data, "__len__"):
231                 data = data[0]
232
233             self.summary.scalar(name=name, data=data, step=step)
234             self.summary_writer[label].flush()
235
236     def write_items(self, name: str, data: dict[str, list[float]], step: int) -> None:
237         for label in data.keys():
238             self.write_item(name, label, data[label], step)
239
240

```

```

241 class Telemetry:
242     def __init__(self) -> None:
243         self.loggers: list[TBWriter] = []
244
245     def add_logger(self, logger: TBWriter) -> None:
246         self.loggers.append(logger)
247
248     def write_item(self, name: str, label: str, data: Any, step: int) -> None:
249         for logr in self.loggers:
250             logr.write_item(name, label, data, step)
251
252     def write_items(self, name: str, data: dict[str, list[float]], step: int) -> None:
253         for logr in self.loggers:
254             logr.write_items(name, data, step)

```

12.4 utilities/planning.py

```
1  # Copyright (c) 2024 Mobius Logic, Inc.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #   http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from .timelines import Timeline, Timelines, TimelineEvent
16 import numpy as np
17 from numpy.random import Generator, PCG64DXSM
18 import copy
19 import pandas as pd
20
21 class State:
22     def __init__(self, parameters, seed, trajectory, current_t=0):
23         self.parameters = parameters
24         self.seed = seed
25         self.trajectory = trajectory
26         self.current_t = current_t
27         self.cumulative_rews = dict()
28
29     def __repr__(self):
30         return self.__str__()
31
32     def __str__(self):
33         return f"parameters: {self.parameters}, seed: {self.seed}, trajectory len: {len(self.trajectory)}"
34
35
36 class Trajectory:
37     def __init__(self, env, initial_return, intial_frame=None):
38         self.players = copy.copy(env.possible_agents)
39         self.trajectory = []
40         self.actions = []
41         self.step_returns = []
42         self.agent_info = []
43         self.add(
44             env, None, initial_return, None
45         )
46         self.frames = []
47         if intial_frame is not None:
48             self.frames.append(intial_frame)
49
50     def add(self, env, action, step_return, agent_info, frame=None):
51         self.trajectory.append(
52             {
53                 "action": action,
54                 "step_return": step_return,
55                 "agent_info": agent_info
56             }
57         )
58         self.actions.append(action)
59         self.step_returns.append(step_return)
60         self.agent_info.append(agent_info)
61
62         if frame is not None:
63             self.frames.append(frame)
64
65     def __getitem__(self, key):
66         return self.trajectory[key]
67
68     def __len__(self):
69         return len(self.trajectory)
70
71
72 ## Telemetry Object:
73 class Telemetry:
74     def __init__(self, name, **kwargs):
75         self.data = None
76         self.data_labels = None
77         self.xscale = None
```

```

78         self.title = None
79         self.name = None
80         self.xlabel = None
81         self.ylabel = None
82         self.colors = None
83
84         for k,v in kwargs.items():
85             if k in self.__dict__.keys():
86                 self.__dict__[k] = v
87             else:
88                 raise Exception(f"Keyword argument {k} not in telemetry class.")
89
90     def __str__(self) -> str:
91         shape = getattr(self.data, "shape", None)
92         return f"Telemetry - name: {self.name}, shape: {shape}"
93
94     def __repr__(self) -> str:
95         return self.__str__()
96
97     def set_data(self, data):
98         self.data = data
99         self.data_labels = [""] * data.shape[0]
100        self.colors = [None] * data.shape[0]
101
102    def as_df(self):
103        # print("Telemetry:", self.data, self.data_labels, self.xscale)
104
105        tmp = pd.DataFrame(self.data.T)
106
107        # print("Telemetry:", tmp)
108        columns = copy.copy(self.data_labels)
109        for i in range(len(columns)):
110            if columns[i] is None:
111                if self.name is not None:
112                    columns[i] = self.name
113                else:
114                    columns[i] = ""
115
116        tmp.columns = columns
117
118        # print("Telemetry:", tmp)
119        tmp.index = self.xscale
120        return tmp
121
122    # Returns a fixed seed and fixed set of parameters
123    class BaseParameterGenerator:
124        def __init__(self, seed, param):
125            self.seed = seed
126            self.param = param
127
128        def sample(self):
129            return (self.seed, self.param)
130
131        def setseed(self, seed):
132            pass
133
134        def update_param(self, param):
135            self.param = param
136
137        def freeze(self):
138            pass
139
140        def thaw(self):
141            pass
142
143    # This returns a random seed but a fixed set of parameters
144    class SeededParameterGenerator:
145        def __init__(self, seed, param):
146            self.np_random = Generator(PCG64DXSM(seed=seed))
147            self.param = param
148            self._frozen = False
149
150        def sample(self):
151            if self._frozen:
152                return (self._frozen_random, self.param)
153            else:
154                return (self.np_random.integers(low=2**32), self.param)
155
156        def setseed(self, seed):
157            self.np_random = Generator(PCG64DXSM(seed=seed))
158

```

```

159     def update_param(self, param):
160         self.param = param
161
162     def freeze(self):
163         self._frozen = True
164         self._frozen_random = self.np_random.integers(low=2**32)
165
166     def thaw(self):
167         self._frozen = False
168
169     # %%
170     class CommunicationSchedule(Timelines):
171         @staticmethod
172         def repeating(
173             checkin_frequency=1,
174             checkin_offset=0,
175             blackout_frequency=1,
176             blackout_length=0,
177             blackout_offset=0,
178             allow_agent_break=True,
179         ):
180             ## This is going to secretly overload the standard timeline behavior to make it
181             ## infinite
182             tmp = CommunicationSchedule(length=1, allow_agent_break=allow_agent_break)
183             tmp.__dict__["checkin_frequency"] = checkin_frequency
184             tmp.__dict__["checkin_offset"] = checkin_offset
185             tmp.__dict__["blackout_frequency"] = blackout_frequency
186             tmp.__dict__["blackout_length"] = blackout_length
187             tmp.__dict__["blackout_offset"] = blackout_offset
188             tmp.__dict__["allow_agent_break"] = allow_agent_break,
189             tmp.__dict__["repeating"] = True
190
191             blackout_event = TimelineEvent(parameters=None, label="blackouts")
192             checkin_event = TimelineEvent(parameters=None, label="checkins")
193
194             def next_event(time):
195                 t1 = (checkin_frequency - time + checkin_offset) % checkin_frequency
196                 if t1 == 0:
197                     t1 = checkin_frequency
198                 return t1 + time, checkin_event
199
200             tmp.checkins.__dict__["next_event"] = next_event
201
202             def get(time, role):
203                 if (time - blackout_offset) % blackout_frequency < blackout_length:
204                     return True
205
206                 return False
207
208             tmp.blackouts.__dict__["get"] = get
209
210             return tmp
211
212     @staticmethod
213     def from_lists(length, blackouts = [], checkins = [], allow_agent_break=True):
214         tmp = CommunicationSchedule(length, allow_agent_break)
215         blackout_event = TimelineEvent(parameters=None, label="blackouts")
216         checkin_event = TimelineEvent(parameters=None, label="checkins")
217
218         for t in blackouts:
219             tmp.blackouts.add_events(t, blackout_event)
220
221         for t in checkins:
222             tmp.checkins.add_events(t, checkin_event)
223
224         return tmp
225
226     def __init__(self, length, allow_agent_break=True):
227         super().__init__(labels=["blackouts", "checkins"], length=length)
228         self.ALLOW_AGENT_BREAK = allow_agent_break
229         self.repeating = False
230
231     def get(time, role):
232         return False
233
234     self.blackouts.__dict__["get"] = get
235     # self.checkins.add_event(time=0, event=TimelineEvent(parameters=None, label="checkin"))
236
237     def __getstate__(self):
238         return self.__dict__

```

```

240
241     def __setstate__(self, state):
242         self.__dict__ = state
243
244
245     def __deepcopy__(self, memo):
246         if self.repeating:
247             return CommunicationSchedule.repeating(
248                 checkin_frequency=copy.copy(self.checkin_frequency),
249                 checkin_offset=copy.copy(self.checkin_offset),
250                 blackout_frequency=copy.copy(self.blackout_frequency),
251                 blackout_length=copy.copy(self.blackout_length),
252                 blackout_offset=copy.copy(self.blackout_offset),
253                 allow_agent_break=copy.copy(self.allow_agent_break),
254             )
255
256         tmp = self.__class__(labels=[])
257         tmp.timelines = copy.deepcopy(self.timelines)
258
259         return(tmp)
260
261 class COA(Timeline):
262     def to_dict(self):
263         d = {
264             time: {event.label: event.to_dict() for event in events} for time, events in
265                 self.timeline.items()
266         }
267         return d
268
269     def get_actions(self, t):
270         return self.timeline.get(t, default=None)

```


12.5 utilities/timelines.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 import numpy as np
16 import matplotlib.pyplot as plt
17 import json
18 import copy
19
20 from typing import Any
21
22
23 ## Aux Classes:
24 # Note: Everything here has been written so that it (at present)
25 # deepcopies. If you do anything to make a model not deepcopy
26 # by default you'll need to add a __deepcopy__ function
27
28 # %%
29 #####
30 class TimelineEvent:
31     id_idx = 0
32     @staticmethod
33     def from_dict(event):
34         tmp = TimelineEvent(
35             event["label"],
36             event["parameters"],
37             tags = {k:v for k,v in event.items() if (k not in {"label", "parameters", "id"})},
38             id = event.get("id", None)
39         )
40
41         return tmp
42
43     def __init__(self, label, parameters, tags: dict = dict(), id=None):
44         if id == None:
45             self.id = TimelineEvent.id_idx
46             TimelineEvent.id_idx += 1
47
48         self.label = label
49         self.parameters = parameters
50         self.tags = tags
51
52     def to_dict(self):
53         return {
54             "label": self.label,
55             "parameters": self.parameters,
56             "tags": self.tags
57         }
58
59     def __repr__(self) -> str:
60         return str(self)
61
62     def __str__(self) -> str:
63         return f"label: {self.label}, params: {self.parameters}, tags: {self.tags}"
64
65     def copy(self):
66         return copy.copy(self)
67
68
69 class TimelineInterval(TimelineEvent):
70     def __init__(self, *args, extent=1):
71         super().__init__(*args)
72         self.extent = extent
73         self.range = range(self.start, self.start + extent + 1)
74
75
76 class Timeline:
77     def __init__(self, length=0, fixedlength=None):
78         self.timeline = dict()
```

```

79         self.events = dict()
80         self.labels = dict()
81
82         self.fixedlength = False
83         self.length = length
84         self.allow_dup_labels = False
85
86         if fixedlength is not None:
87             if length>0:
88                 raise Exception("Both length and fixedlength are specified. Only one should be.")
89             self.fixedlength = True
90             self.length = fixedlength
91
92     def __repr__(self) -> str:
93         return f"Timeline: {str(self.events)}"
94
95     def __len__(self):
96         return self.length
97
98     def __deepcopy__(self, memo):
99         return(copy.deepcopy(self.__dict__))
100
101     def add_events_from_dict(self, events):
102         # Expected format:
103         # event = {
104         #     "start": int,
105         #     "action": str,
106         #     "parameters": np.array,
107         #     "role": str,
108         # }
109
110         for event in events:
111             self.add_event(event["start"], TimelineEvent.from_dict(event))
112
113
114     def add_events(self, events):
115         if type(events) is TimelineEvent:
116             self.add_event(events)
117         else:
118             for t, e in events:
119                 self.add_event(t, e)
120
121     def add_event(self, time, event: TimelineEvent):
122         if self.fixedlength:
123             if time > self.length:
124                 raise Exception(f"Timeline has fixed length {self.length} but event added at {time}.")
125         else:
126             self.length = max(self.length, time)
127
128         if not self.allow_dup_labels:
129             mark_for_delete = []
130             for e in self.timeline.get(time, []):
131                 if e.label == event.label:
132                     mark_for_delete.append(e)
133
134             for e_del in mark_for_delete:
135                 self.delete_event_at(time=time, event=e_del)
136
137
138         if time not in self.timeline.keys():
139             self.timeline[time] = set()
140
141         if event not in self.events.keys():
142             self.events[event] = set()
143
144         if event.label not in self.labels.keys():
145             self.labels[event.label] = set()
146
147         self.timeline[time].add(event)
148         self.timeline = dict(sorted(self.timeline.items()))
149
150         self.events[event].add(time)
151         self.labels[event.label].add((time, event))
152
153     def delete_event_at(self, event, time):
154         self.events[event].remove(time)
155         if len(self.events[event]) == 0:
156             del self.events[event]
157
158         self.timeline[time].remove(event)
159         if len(self.timeline[time]) == 0:

```

```

160         del self.timeline[time]
161         if time == self.length-1 and not self.fixedlength:
162             self.length = max(self.timeline.keys())
163
164         self.labels[event.label].remove((time, event))
165         if len(self.labels[event.label]) == 0:
166             del self.labels[event.label]
167
168     def delete_all_occurences(self, event):
169         while event in self.event.keys():
170             t = list(self.events[event])[0]
171             self.delete_event_at(event=event, time=t)
172
173     def move_events(self, time, to, events=None):
174         if events is None:
175             events = copy.copy(self.timeline[time]) # we're modifying the object as we iterate through it
176
177         for event in events:
178             self.delete_event_at(event = event, time=time)
179             self.add_event(event=event, time=to)
180
181
182
183     def add_interval(self, event: TimelineInterval):
184         for t in event.range:
185             self.add_event(t, event)
186
187     def first(self, event):
188         return(min(self.events[event]))
189
190     def last(self, event):
191         return(max(self.events[event]))
192
193     def interval_start(self, event, t):
194         if t not in self.events[event]:
195             return None
196
197         while t in self.events[event]:
198             t += -1
199
200         return t + 1
201
202     def interval_end(self, event, t):
203         if t not in self.events[event]:
204             return None
205
206         while t in self.events[event]:
207             t += 1
208
209         return t - 1
210
211     def get_all_labeled(self, label):
212         return self.labels[label]
213
214     def get_events_at_time(self, time):
215         return self.timeline[time]
216
217     def get(self, time, label):
218         r = []
219         if label not in self.labels.keys():
220             raise Exception(f"'{label}' not found in known labels: {list(labels.keys())}")
221
222         for t, event in self.labels[label]:
223             if t == time:
224                 r.append(event)
225
226         if len(r) == 1:
227             return r[0]
228
229         if len(r) == 0:
230             return None
231
232         if len(r) > 1:
233             raise Exception(f"Multiple events with same label at same time: {r}")
234
235     def itterate_over_label(self, label):
236         tmp = dict()
237         for time, event in self.labels[label]:
238             if time not in tmp.keys():
239                 tmp[time] = []
240

```

```

241         tmp[time].append(event)
242
243     # Returns a key sorted list of tuples (t, [e1, e2,...])
244     return iter(sorted(tmp.items()))
245
246 def next_event(self, time):
247     L = list(self.timeline.keys())
248     if len(L) == 0:
249         return None, None
250
251     last_high = L[-1]
252     if time >= last_high:
253         return None, None
254     if time < L[0]:
255         return L[0], self.timeline[L[0]]
256
257     # Go through timeline via bisection. Probably overkill
258     while len(L)>1:
259         half_idx = int(len(L)/2)
260
261         if L[half_idx] > time:
262             last_high = L[half_idx]
263             L = L[:half_idx]
264         else:
265             L = L[half_idx:]
266             if L[-1] > time:
267                 last_high = L[-1]
268
269     return last_high, self.timeline[last_high]
270
271 def __iter__(self):
272     ## Should already be sorted
273     return iter(sorted(self.timeline.items()))
274
275 def add_timeline_to(self, timeline, allow_dup_labels=True):
276     ## Add new coa elements
277     for event, times in timeline.events.items():
278         if event not in self.events.keys():
279             self.events[event] = set()
280
281         to_update = times - self.events[event]
282
283         for t in to_update:
284             self.add_event(time=t, event=event)
285
286 def display(self):
287     PADDING = 3
288     LINEWIDTH = 64
289     LABELLENGTH = 8
290     TLSPACE = LINEWIDTH - PADDING - LABELLENGTH
291
292     if self.length < TLSPACE:
293         step = 1
294         LINEWIDTH = LABELLENGTH + self.length
295         TLENGTH = self.length
296     else:
297         step = int(np.ceil(self.length/(TLSPACE)))
298         TLENGTH = int(np.ceil(self.length/step))
299
300     ## Time:
301     ticklabs = [" "]*LINEWIDTH
302     ticklabs[0:5] = list("Time:")
303     ticks = [" "]*LABELLENGTH + ["-"]*(TLENGTH+1)
304
305     demarks = list(range(0,self.length, step*10))
306
307     for j, i in enumerate(demarks):
308         j = LABELLENGTH + j*10
309
310         c = len(str(i))
311         tmp = [" "]*7
312         if c < 8:
313             tmp[(3-int((c-1)/2)):(3-int((c-1)/2)) + c] = str(i)
314         else:
315             tmp = "{:.2E}".format(c)
316
317         ticklabs[j-3:j+4] = tmp
318
319         ticks[j] = "|"
320
321     ticks[-1] = "|"

```

```

322
323     ## Lines
324
325     lines = {
326         label: list(str(label)[:LABELLENGTH] + [" "] * max(0, LABELLENGTH - len(label)) +
327                     [" "] * TLENGTH
328                     for label in self.labels.keys())
329     }
330
331     for label, events in self.labels.items():
332         for t, event in events:
333             lines[label][LABELLENGTH + int(t/step)] = "o"
334
335     lines = [" ".join(line) for line in lines.values()]
336     lines.insert(0, " ".join(ticklabs))
337     lines.insert(1, " ".join(ticks))
338
339     return "\n".join(lines)
340
341 class Timelines:
342     def __init__(self, labels, length=0, fixedlength = None):
343         self.timelines = {label: Timeline(length=length, fixedlength=fixedlength) for label in labels}
344
345     def get_events(self, time):
346         return {label: tl.get_events_at_time(time) for label, tl in self.timelines.items()}
347
348     def __repr__(self) -> str:
349         return str(self.timelines)
350
351     def __getattr__(self, __name):
352         return self.timelines[__name]
353
354     # def __deepcopy__(self, memo):
355     #     tmp = self.__class__(labels=[])
356     #     tmp.timelines = copy.deepcopy(self.timelines)
357
358     #     return(tmp)
359
360     def __getstate__(self):
361         return self.timelines
362
363     def __setstate__(self, d):
364         self.timelines = d

```

12.6 utilities/tools.py

```
1 # Copyright (c) 2024 Mobius Logic, Inc.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 from typing import Any
16
17
18 class MultiDict:
19     def __init__(self, tags: dict = dict()):
20         self.tags = tags # tags - str(type): list(values)
21         self._vals = dict()
22         self._refs = {tag: {val:[] for val in values} for tag, values in tags.items()}
23
24     def __getattr__(self, __name: str) -> Any:
25         if __name in self.tags.keys():
26             return SliceMakerDict(__name, self)
27
28     def __getitem__(self, select_name: dict):
29         new_tags = dict(self.tags)
30
31         for tag_name in select_name.keys():
32             del new_tags[tag_name]
33
34         new_mdct = MultiDict(new_tags)
35
36         items = [set(self._refs[tag_name][tag]) for tag_name, tag in select_name.items()]
37         items = set.intersection(*items)
38
39         for item in items:
40             new_tag = dict(self._vals[item])
41             for tag_type_name in select_name.keys():
42                 del new_tag[tag_type_name]
43
44             new_mdct[new_tag] = item
45
46         return new_mdct
47
48
49     def __setitem__(self, key: dict, value: Any):
50         # Following are sufficient to check all keys supplied:
51         if not len(key.keys()) == len(self.tags.keys()):
52             raise Exception(f"Mismatched key lengths between {key.keys()} and {self.tags.keys()}")
53
54         for tag_type, tag in key.items():
55             try:
56                 tags = self._refs[tag_type]
57             except:
58                 raise Exception(f"Trying to set tag_type {tag_type} but this is not found in
59                                     {self.tags.keys()}")
60
61             if tag not in tags:
62                 self.tags[tag_type].append(tag)
63                 tags[tag] = list()
64
65             tags[tag].append(value)
66
67             self._vals[value] = key
68
69     def __repr__(self) -> str:
70         s = ""
71         tab_count = []
72         for tag_type in self.tags.keys():
73             count = int(np.floor(len(tag_type) / 8)) + 1
74             tab_count.append(count)
75             s += f"{tag_type}\t"
76
77         s += "value: \n"
```

```

78         for item, tag in self._vals.items():
79
80             # Keep the same order as above
81             for i, tag_type in enumerate(self.tags.keys()):
82                 max_len = tab_count[i]*8 - 2
83                 tag_len = len(str(tag[tag_type]))
84
85                 if tag_len > max_len:
86                     sp = str(tag[tag_type])[tab_count[i]*8-2] + "\t"
87                 else:
88                     count = int(np.floor((max_len - tag_len)/8) + 1)
89                     sp = str(tag[tag_type]) + "\t"*count
90
91                 s += sp
92
93             s += str(item)
94             s += "\n"
95
96         return s
97
98     class SliceMakerDict:
99         def __init__(self, tag_type_name, parent: MultiDict):
100             self.parent: MultiDict = parent
101             self.tag_type_name = tag_type_name
102
103         def __repr__(self) -> str:
104             return str(self.parent.tags[self.tag_type_name])
105
106         def __getitem__(self, key):
107             new_tags = dict(self.parent.tags )
108             del new_tags[self.tag_type_name]
109
110             new_mdct = MultiDict(new_tags)
111             for item, tag in self.parent._vals.items():
112                 if tag[self.tag_type_name] == key:
113                     new_tag = dict(tag)
114                     del new_tag[self.tag_type_name]
115
116                     new_mdct[new_tag] = item
117
118             return new_mdct
119
120
121
122
123
124     def unit_vector(vector):
125         """Returns the unit vector of the vector."""
126         return vector / np.linalg.norm(vector)
127
128
129     def angle_between(v1, v2):
130         """Returns the angle in radians between vectors 'v1' and 'v2':
131
132         >>> angle_between((1, 0, 0), (0, 1, 0))
133         1.5707963267948966
134         >>> angle_between((1, 0, 0), (1, 0, 0))
135         0.0
136         >>> angle_between((1, 0, 0), (-1, 0, 0))
137         3.141592653589793
138         """
139         v1_u = unit_vector(v1)
140         v2_u = unit_vector(v2)
141         return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))
142
143
144     def plot_traj(p, q, v, traj=None, w=[0, 0], ax=None):
145         if ax is None:
146             ax = plt.gca()
147         if not traj is None:
148             ax.plot(traj[:, 0], traj[:, 1], "--")
149             ax.plot(p[0], p[1], "or")
150             ax.plot(q[0], q[1], "og")
151             ax.quiver(p[0], p[1], v[0], v[1], angles="xy", scale_units="xy", scale=1)
152             ax.quiver(
153                 p[0], p[1], float(w[0]), float(w[1]), angles="xy", scale_units="xy", scale=1
154             )
155             # ax.set_xlim(-60,10)
156             # ax.set_ylim(-10,10)
157         return ax
158

```

```

159
160 def plot_traj_3d(p, q, v, traj=None, w=[0, 0], ax=None):
161     if ax is None:
162         ax = plt.figure().add_subplot(projection="3d")
163     if not traj is None:
164         ax.plot(traj[:, 0], traj[:, 1], traj[:, 2], "--")
165     ax.plot(p[0], p[1], p[2], "or")
166     ax.plot(q[0], q[1], q[2], "og")
167     ax.quiver(p[0], p[1], p[2], v[0], v[1], v[2])
168     ax.quiver(p[0], p[1], p[2], float(w[0]), float(w[1]), float(w[2]))
169     return ax
170
171
172 def r(t, w, p, v):
173     return p + (t**2) * w / 2 + t * v
174
175
176 def r_p(t, w, v):
177     return t * w + v
178
179
180 def g(t, w, t1, p, v):
181     if t < t1:
182         return r(t, w, p, v)
183     else:
184         return r(t1, w, p, v) + (t - t1) * r_p(t1, w, v)
185
186
187 def dg(t, w, t1, p):
188     return r(t1, w, p) + t * r_p(t1, w, p)
189
190
191 def rotation_matrix(theta):
192     theta
193
194     rotMatrix = np.array(
195         [[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]]
196     )
197
198     return rotMatrix

```