Contents

1	/		3
•	1.1	README.md	3
	1.2	agents.py	5
	1.3		0 01
	$1.3 \\ 1.4$		18
	$\frac{1.4}{1.5}$		10 19
	_	1 0	
	1.6		24
	1.7	·	25
	1.8	texify.py	26
9	DAG	CIT O	
2	DAS		28
	2.1	11 10	28
	2.2		34
	2.3	The state of the s	11
	2.4	html_objects.py	12
3	DAG	SH/Documentation 4	16
0	3.1		16
	3.2	network_view.py	18
1	DAG	m SH/assets	60
•			50
	4.1	typography.css	,0
5	exai	mples/MAInspection 5	63
	5.1	Readme.md	
	5.2		54
	5.3	11 10	56
	5.4		59
	0.4	train-direction.py	,,,
6	exai	mples/MAInspection/Environments 6	32
	6.1		32
	6.2		34
	6.3		39
	0.0		,,
7	exai	mples/MAInspection/Experiment 8	35
	7.1	dash_app.yaml	35
	7.2	dash_app_template.yaml	36
	7.3		37
	7.4		38
8	exai	mples/MAInspection/assets 8	39
	8.1	typography.css	39
9	exai	1 / 1 0)2
	9.1		92
	9.2		93
	9.3	0 10	95
	9.4	train_director.py	98
10		mples/Multipolicy/Experiment 10	
		dash_app.yaml	
	10.2	dash_app_template.yaml)2
	10.3	train_agents.yaml)3
		train director vaml	

l examples/Multipolicy/assets			
1.1 typography.css	105		
tilities	108		
2.1 PZParams.py	108		
2.2 PZWrapper.py	110		
2.3 iotools.py			
2.4 planning.py	117		
2.5 timelines.py	121		
2.6 tools.py	126		

1 /

1.1 README.md

Controller-Agnostic Orchestration Architecture (COACH)

```
## A Framework for Training Semi-Autonomous Agents and Directors Leveraging Existing Simulations
2
3
   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 enviornemnt 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 raining director agents. In
        addition, it provides a prefab agents and an interactive plan visualization system using DASH.
6
   This library can be used as is, but has been deisgned as a starting point for interactive planning
        problems.
7
    <img src="examples/Multipolicy/Interface.png">
8
9
10
   ## Termanology:
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 understand 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]
   * Agent - A piece of code that takes observations and returns actions.
   * Actor - An agent assigned to a specific role in multiagent simulation, and equiped with an instruction
15
        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.
   * 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 depenant, or agent
        dependant with blackout conditions), and the courses of action may be updated.
    * Director - An agent that takes observations recorded by actors, and generates a courses of action.
    * Trajectory - Action/observation/rewards for all agents over a period of time.
22
23
   # Getting started
25
   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.
26
    ## Examples:
29
    ### Multiploicy Director
30
31
    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.
32
   In this example we see how to train policies on a PettingZoo environment, use those policies and that
33
        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.
34
    Finally, we construct a Dash app to display our solution.
35
36
37
    This example uses all most entirely default functionality. For more advanced functionality see the
        MAInspection example.
38
39
    ### Multiagent Inspection
40
41
    This example shows how to solve a simple satelite inspection problem in two steps:
42
    * 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.
    * After this "waypointer"
43
    agent is trained the director is then trained to select waypoints that easily solve the problem.
44
45
46
   This example also demonstrates how to create custom agents, a custom coach environment, custom
```

```
47
    ## Overview of Environment Files:
48
49
    * `coach.py` - Contains the class that wraps a PZ env, turning it into a planning environment.
50
51
    COACHEnvironemnt (
52
53
               env creator: callable.
               \verb"parameter_generator:BaseParameterGenerator",
54
55
               agents=dict().
               fill_random=True.
56
               comm_schedule:Timeline=None,
57
               seed=6234235.
58
               TrajectoryClass = Trajectory,
59
    )
60
61
    * * `env_creator` - A function that creates a parallel PettingZoo env.
62
     * * `parameter_generator` - A parameter generator object. Simulations call the parameter generator for
63
           samples. The BaseParameterGenerator simply returns the same parameters each time, the
           {\tt 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.
    * * 'fill_random' - If 'True' any agent roles not specified above will be set to the RandomActor agent.

* * 'comm_schedule' - A 'utilities.planning.CommunicationsSchedule' object that determines contact times
65
66
           and blackout times for director/agent communication.
    * * `seed` - Random seed.

* * `TrajectoryClass` - Trajectories save agent history. By default all actions and step returns are
67
68
           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
    * * `env_creator` - A function that creates a parallel PettingZoo env. * * `COACHEnvClass` - COACHEnvironemnt class that will be wrapped.
74
75
76
77
     ## Overview of Auxiliary Files
     * `agents.py` - Library of agents, sample agent classes include RandomActor, DefaultActor and
           SB_PPOPoliciesActor
79 * 'params.py' - Standard MarcoPolo params functionality for evolving parameters
    * 'utilities.iotools.py' - Tools for saving and reloading
* 'utilities.planning.py' - Holds timeline classes specifically for COACH, including
'CommunicationsSchedule' and 'COA'
80
   * `utilities.PZParams.py` - Parameter file for environments from the PettingZoo library
* `utilities.PZWrapper.py` - Wrapper for environments from the PettingZoo library
* `utilities.timelines.py` - Timeline classes
* `utilities.tools.py` - Miscellaneous tools used by parts of the coach env.
83
    st `DASH` - Interactive Course Of Action viewer. See Multipolicy Example for more.
88 * 'examples' - Examples of how to use the coach environment.
```

trajectory classes and custom trajectory visualizations.

1.2 agents.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
2
3
    # Licensed under the Apache License, Version 2.0 (the "License");
      you may not use this file except in compliance with the License.
      You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
    # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
15
   import sys
16
17
    from utilities.timelines import TimelineEvent
    from utilities.planning import COA
18
19
    from numpy.random import PCG64DXSM, Generator
20
    import numpy as np
    import copy
import logging
21
22
23
    from gymnasium.spaces import Box
24
25
    logger = logging.getLogger(__name__)
26
27
    from collections import OrderedDict
    from stable_baselines3 import PPO
28
29
    import sys, inspect
30
    *************************************
31
32
33
    ## Interfaces
34
    ##
    *******************
35
36
37
    class ActionBox(Box):
        def __init__(self, low=[], high=[], shape=[], default=[], description=None):
38
            if len(low)>0:
39
                super().__init__(low=low, high=high, shape=shape)
40
41
42
                self.low = []
43
                self.high = []
44
            self.description = description
45
46
47
            if len(default) == len(low):
                self.default = default
48
49
             elif len(default) == 0:
50
                default = []
51
                for 1, h in zip(low, high):
52
                     # If the bounds are finite, take the average.
53
                     if np.isfinite(1) and np.isfinite(h):
54
                         default.append((1+h)/2)
55
                     # If one is infinte, 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):
                         default.append(1)
                     ## Otherwise, assume 0
60
                     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 = []
            for i in range(len(self.low)):
73
74
                actions.append(
                     ActionBox(
75
76
                         low = np.array([self.low[i]])
                         high = np.array([self.high[i]]),
shape = (1,),
77
```

```
79
                          description = self.description[i]
 80
                      )
 81
                 )
 82
 83
             return actions
 84
         def equals(self, other):
    # All defualt interfaces are the same
 85
 86
 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
             if not all([a==b for a,b in zip(self.shape, other.shape)]):
 93
 94
                 return False
 95
             if not all([a==b for a,b in zip(self.default, other.default)]):
 96
                  return False
 97
             return True
98
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
             self.name = self.__class__.__name__
111
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):
              # Return the dctionary of possible actions with their parameter
121
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, espeaclly if the interface takes parameters.
         def equals(self, other):
127
             # All defualt interfaces are the same
128
             if type(other) is type(self):
129
130
                 return True
131
             return False
132
133
     class DefaultInterface(TrivialInterface):
134
         def __init__(self, role, env, max_action_len=None):
    super().__init__(role, env)
135
136
137
138
             if not max_action_len:
139
                  max_action_len = np.inf
140
             # The burn is "legnth" + "action space"
141
             burn = ActionBox(
142
                  low=np.concatenate([(0,), self.env_action_space.low]),
143
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
             self.action_dictionary = {"burn": burn}
149
150
151
         def equals(self, other):
152
              # All defualt 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):
```

```
def __init__(self, role, env):
160
             super().__init__(role, env)
161
162
163
             self.action_dictionary = {"random": ActionBox()}
164
165
     class SBPolicyInterface(TrivialInterface):
166
         def __init__(self, role, env, n_policies, max_action_len):
167
             super().__init__(role, env)
168
169
             one_hot_low = [0]*n_policies
170
             one_hot_high = [1]*n_policies
171
             desc = ["Action Length"] + [f"policy_{i}" for i in range(n_policies)]
172
173
             policies = ActionBox(
174
                 low=np.array([0] + one_hot_low),
175
176
                 high=np.array([max_action_len] + one_hot_high),
177
                 shape=(len(one_hot_low) + 1,),
178
                 description=desc
179
180
             self.action_dictionary = {"Policies": policies}
181
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
         def get_action(self, obs, t, mean_mode=False):
    return self.none_action, {"acting": False, "coa_done": True}
215
216
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:
                 if not self.interface.equals(self.reference_interface):
    raise Exception("Actor interface does not match reference interface.")
229
230
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():
                 e = self.coa.get(time = t, label = "burn")
252
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)
                 last_duration = self.coa.get(time = last_t, label= "burn").parameters[0]
263
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
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):
             super().__init__(role)
self.seed = seed
289
290
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
                 {\tt low=self.interface.action\_space.low}, \ {\tt high=self.interface.action\_space.high}
301
302
             return r, {"acting": True, "coa_done": False}
303
304
305
         def reset(self, env):
306
             super().reset(env)
307
308
             self.interface = RandomActor.InterfaceType(self.role, env)
             self.np_random = Generator(PCG64DXSM(seed=self.seed))
309
310
311
     class SB_PPOPoliciesActor(BasicActor):
312
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
                   if model_path[-3:] == "zip":
323
                   model_path = model_path.split(".")[0]
self.policies[policy] = PPO.load(model_path)
324
325
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
              acting = False
334
              coa_done = False
335
336
337
              if t in self.coa.timeline.keys():
338
                   ## Start new policy
                   e = self.coa.get(time = t, label = "Policies")
339
                   self.time_remaining = e.parameters[0]*self.max_action_len
logger.debug("Agent: start new policy %s", e) # DEBUG
340
341
                   logger.debug("policy: %s, timeleft: %s",
342
                        self.policy_names[int(np.argmax(e.parameters[1:]))], self.time_remaining) # DEBUG
343
                   self.current_policy = self.policy_names[int(np.argmax(e.parameters[1:]))]
344
345
              # self.next_waypoint_abs = np.array([7.08763559, -25.61246231, -167.51736098])
346
347
              if (self.current_policy is not None) and (self.time_remaining > 0):
348
                   acting = True
349
                   action, _ = self.policies[self.current_policy].predict(obs, deterministic=True)
350
                   self.time\_remaining += -1
351
352
                   action = copy.copy(self.none_action)
353
354
              if self.time_remaining == 0:
355
                   coa_done = True
356
357
              return action, {
358
                   "acting": acting,
"coa_done": coa_done,
359
360
                   "bad_command": bad_command,
361
              }
362
363
          def reset(self, env):
364
              super().reset(env)
365
              self.time_remaining = -1
366
              self.current_policy = None
367
368
              self.interface = SB_PPOPoliciesActor.InterfaceType(
369
                   self.role,
370
                   env.
371
                   n_policies=len(self.policy_names),
372
                   max_action_len=self.max_action_len
373
374
375
              if self.reference_interface is not None:
                   if not self.interface.equals(self.reference_interface):
376
377
                       raise Exception("Actor interface does not match reference interface.")
378
379
380
     classes = [
381
          cls_obj
          for cls_name, cls_obj in inspect.getmembers(sys.modules[__name__]) if inspect.isclass(cls_obj) and cls_obj.__module__ == __name__
382
383
384
     1
385
     Interfaces = {}
386
     for c in classes:
387
388
          # BasicActor
          if issubclass(c, TrivialInterface):
    Interfaces[c] = []
389
390
391
392
     Agents = {}
     for c in classes:
393
394
          # BasicActor
          if issubclass(c, BasicActor):
395
396
              Interfaces[c.InterfaceType].append(c)
397
              Agents[c.__name__] = c
```

1.3 coach.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
2
3
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
      You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
  # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
15
   from typing import Any
   import numpy as np
16
    import copy
17
    import logging
18
    logger = logging.getLogger(__name__)
19
   import argparse
20
21
    import matplotlib.pyplot as plt
22
   from numpy.random import PCG64DXSM, Generator
23
24
25
    import agents as agents
26
    from agents import (
        BasicActor,
27
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,
        State,
38
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
            arguments that were passed to the 'main()' function
55
56
57
58
        the class to use in creating env objects """
60
62
63
        return COACHEnvironment
64
65
    class COACHEnvironment:
66
        agent_selection = agents
67
68
        def __init__(
69
            self.
70
             env_creator: callable,
71
             \verb"parameter_generator:BaseParameterGenerator",
72
             agents=dict().
73
            fill_random=True,
74
             comm_schedule: Timeline=None,
             seed=6234235,
75
            TrajectoryClass = Trajectory,
76
77
             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:
                   comm_schedule = Timelines(labels=["blackout", "checkins"])
 88
 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 environments possible actors"
100
101
102
          {\tt def} \ \_\_{\tt deepcopy}\_\_({\tt self} \ , \ {\tt 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,
                   tmp_agents,
108
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:
                  fill_random = self.fill_random
119
120
121
              self.action_spaces = {}
122
              self.observation_spaces = {}
123
124
              self.agents = dict()
125
126
              for agt in self.env.possible_agents:
                  if agt in self.stored_agents.keys():
    self.agents[agt] = self.stored_agents[agt]
127
128
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)
                   self.action_spaces[agt] = self.agents[agt].interface.action_dictionary
136
                   self.observation_spaces[agt] = self.agents[agt].interface.observation_space
137
138
139
          def possible_models(self):
              default = BasicActor, DefaultActor, RandomActor return {role: default for role in self.env.possible_agents}
140
141
142
143
          def set_coa(self, coas):
              for agt, coa in coas.items():
    self.agents[agt].update_coa(coa)
144
145
146
147
          def get_coa(self):
              return {role: agt.coa for role, agt in self.agents.items()}
148
149
150
          def simulate(
151
              self,
              coas=None.
152
              parameters=None,
153
154
              agents=None,
155
              state=None.
156
              time_steps=None,
157
              comm_steps=None,
158
              render=False
159
          ):
```

```
if comm_steps and time_steps:
160
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)
             if agents is not None:
167
168
                  tmp_env.stored_agents = agents
169
170
             if render:
                  tmp_env.start_rendering()
171
172
              # Set new env params:
173
              if parameters:
174
175
                  tmp_env.augment(parameters)
176
             tmp_env.reset()
177
178
             tmp_agts = tmp_env.agents
179
180
             if state:
181
                  if parameters:
182
                      logger.info(
                           "Note: State parameters will overwrite passed parameters."
183
184
185
                  tmp_env.setstate(state)
186
             # Set new COA for agents
187
188
              if coas:
189
                  for agt, coa in coas.items():
190
                      tmp_agts[agt].update_coa(coa)
191
192
              # Run simulation, recording trajectory
              if time_steps:
193
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
                      truncs = list(tmp_env.step_return[-1][3].values())
206
                      if all([a or b for a, b in zip(terms, truncs)]):
                          ended = True
207
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
         def start_rendering(self, steps_per_frame = 10):
    self.steps_per_frame = steps_per_frame
217
218
219
              self.rendering=True
220
221
         def stop_rendering(self):
              self.rendering=False
222
223
224
         def augment(self, parameters):
225
              self.parameter_generator = parameters
226
227
         def reset(self):
228
              seed, parameters = self.parameter_generator.sample()
              self.setup_env(seed, parameters)
229
              self.state.cummulative_rews = {agt: 0 for agt in self.env.possible_agents}
230
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
             # Fill agents based on reset env
238
239
              self._fillagents()
240
```

```
## Reset agents, get their initial conditions.
self.agent_info = [
241
242
243
                  {role: agt.reset(env=self.env) for role, agt in self.agents.items()}
244
245
246
              initial_frame = None
              if self.rendering:
   initial_frame = self.env.render()
247
248
249
250
              traj = self.TrajectoryClass(
251
                  self.env,
252
                  initial_return=self.step_return[0],
253
                  intial_frame=initial_frame
254
255
              self.state = State(parameters, seed, traj, current_t=0)
256
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:]:
                  step_return = self.env.step(step["action"])
264
265
                  self.step_return.append(step_return)
266
267
                  self.state.trajectory.add(
                      env=self.env,
268
                       action=step["action"],
269
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
         def step_env(self, steps=1, coas=None):
    self.step_end = {
294
295
                  "coa_done": [],
"term_or_trunc": False,
296
297
                  "steps_reached": False
298
299
300
301
              if coas:
302
                  self.set_coa(coas)
303
              t = self.state.current_t-1
running = True
304
305
306
307
              while running:
308
                  t += 1
                  logger.debug("Env Step: %s", t) # DEBUG
309
310
                  action = dict()
311
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:
                           action[role] = None
agent_info[role] = "not_alive"
320
321
```

```
322
323
                logger.debug("Env Action: %s", action)
324
325
                obs, rwds, terms, truncs, info = self.env.step(action)
326
327
                frame = None
328
                if self.rendering:
                    if t % self.steps_per_frame == 0:
329
330
                        frame = self.env.render()
331
332
                returns = (
                    {role: self.agents[role].process_observations(o, t) for role, o in obs.items()},
333
334
                    {\tt rwds}\,,\ {\tt terms}\,,\ {\tt truncs}\,,\ {\tt info}
335
336
337
                for role, rwd in returns[1].items():
338
                    self.state.cummulative_rews[role] += rwd
339
340
                self.agent_info.append(agent_info)
341
                self.step_return.append(returns)
342
                self.state.trajectory.add(
343
                    env=self.env,
344
                    action=action,
345
                    step_return=returns,
346
                    agent_info=agent_info,
347
                    frame=frame)
348
349
                self.alive = {agt: not (returns[2][agt] or returns[3][agt]) for agt in self.agents.keys()}
350
351
                break_for_new_COA = False
352
353
                if self.comm_schedule.ALLOW_AGENT_BREAK:
354
                    for role in self.agents.keys():
355
                         if self.comm_schedule.blackouts.get(t, role) is not None:
356
                            if self.agent_info[-1][role]["coa_done"]:
357
                                self.step_end["coa_done"].append(role)
358
                                break_for_new_COA = True
359
                                running = False
360
361
                if not any(self.alive.values()):
                    self.step_end["term_or_trunc"] = True
362
363
                    running = False
364
365
                if steps:
366
                    if t >= (self.state.current_t + steps):
367
                        running = False
368
369
            if steps:
370
                if t == self.state.current_t + steps - 1:
371
                    # We reached the end, even if other things would have terminated it
                    self.step_end["steps_reached"] = True
372
373
            374
375
376
    377
378
    # COA Level Interface:
     379
380
381
         def get_traj_from_coas(self, coas, params=None, from_state=None, render=False):
382
             traj = self.simulate(
383
                coas,
384
                parameters=params,
385
                state=from state.
386
                render=render
387
388
             return traj
389
390
         def get_traj_from_plan(self,
391
            plan,
             params=None.
392
393
             from_state=None,
394
             render=False
395
         ):
             coas = {role: coa for role, coa in plan.items()}
396
397
398
             agent_dict = dict()
399
             for role, model_class in plan.model_classes.items():
400
                 agent_dict[role] = model_class(role, **plan.model_params[role])
401
402
             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
         def trajectory_telemetry(
412
413
              self,
414
              traj,
415
              components=None,
416
              labels=None.
              players=None.
417
418
              ao="observations",
              plan=None,
419
420
              env=None.
421
              cmap = None
422
423
424
              if players is None:
                  players = list(traj.step_returns[0][0].keys())
425
426
427
                  default_cycle = plt.rcParams['axes.prop_cycle'].by_key()['color']
428
                  cmap = lambda i: default_cycle[i%len(default_cycle)]
429
430
431
              traj_len = len(traj)
432
             rewards = Telemetry(
    name="Reward",
433
434
                  title="Reward",
435
                  xscale = np.array(range(traj_len)),
xlabel = "Time Step",
436
437
                  ylabel = "Reward",
438
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 intial 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 origional env spun up.
454
         def plot_trajectory_component(
455
                  self,
456
                  traj,
457
                  components=[0],
458
                  labels=None,
                  players=None,
459
460
                  ao="observations",
                  plan=None,
461
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
              if not labels:
473
                  labels = dict()
474
                  create_labels = True
475
476
              if ao == "observations":
477
478
                  data = \{\}
479
                  all_components = False
480
                  if not components:
                       all_components = True
481
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:
                          labels[p] = [f"{p}: {c}" for c in components]
494
495
             if ao == "actions":
496
                  data = {}
497
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:
                               components = list(range(len(np.array(t[p]).reshape(-1))))
506
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
516
                  components = [0]
517
                  data = traj.step_returns
518
                  xs = np.zeros([len(players), len(components), len(data)-1])
519
520
                  for j, t in enumerate(data):
521
                      if j > 0: # The initial state has no reward
522
                          for i, p in enumerate(players):
523
                              xs[i, :, j] = t[1][p]
524
525
             n_axes = sum([s.shape[1] for s in data.values()])
526
              # Setup Figures
527
             w = n_axes^{-1}/2
528
             w_m = w + n_axes % 2
529
530
             f = plt.gcf()
531
              axes = []
532
             for i in range(w):
                  for j in range(2):
533
534
                      axes.append(plt.subplot2grid((w_m, 4), (i, 2 * j), 1, 2))
535
536
             if n_axes % 2:
537
                  axes.append(plt.subplot2grid((w + 1, 4), (w, 1), 1, 2))
538
539
540
             # Get Data
541
             idx = 0
542
543
             for p in players:
                  for j in range(data[p].shape[1]):
544
                      axes[idx].plot(data[p][:,j])
axes[idx].title.set_text(labels[p][j])
545
546
547
                      idx += 1
548
             # 15 = axes[0].legend(
549
                    bbox_to_anchor=(0.5, -0.05), loc="lower center", bbox_transform=f.transFigure
550
551
             # )
552
             {\tt plt.subplots\_adjust(left=0.1, right=0.9, hspace=0.3, wspace=0.5)}
553
554
555
             fig = plt.gcf()
              if render_mode == "matplotlib":
556
557
                 return fig
558
559
              canvas = FigureCanvas(fig)
560
              canvas.draw()
561
562
             data = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
563
              data = data.reshape(fig.canvas.get_width_height()[::-1] + (3,))
564
             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)
                 return fig
572
573
574
575
576
577
     578
579
     ## Example Usage
     580
581
     if __name__ == "__main__":
582
583
         {\tt from\ examples.MAInspection.env\ import\ MultInspect}
584
         ## Environemntal Parameters
env_params = {"_OBS_REWARD":.01, "num_deputes": 3}
585
586
587
         # Actor Parameters
588
         COACH_params = {
   "Agents": {
589
590
                 "player_0": {
591
                     "class_name":"DefaultActor"
592
593
                     # params: {"parameter": "value"}
                                                        # If the agent has setup parameters
594
595
                 "player_1": {"class_name":"RandomActor"},
                 "player_2": {"class_name":"DefaultActor"},
596
597
             }
598
         }
599
600
         # Create actors from parameters
         agent_dict = dict()
if "Agents" in COACH_params.keys():
601
602
             for role, agent in COACH_params["Agents"].items():
603
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
        # Wrap parameters in parameter generator
parameter_generator = SeededParameterGenerator(23423, env_params)
610
611
612
613
         env = COACHEnvironment(
614
             env_creator = MultInspect,
             default_parameters = parameter_generator,
615
616
             comm_schedule = comms,
             fill_random = True,
617
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():
                 print(f"{role}\t {actions}")
628
629
             input("Press Enter For Next Observation")
```

1.4 directors.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
 2
 3
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
  # Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # 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
    class SB3 PPO Director:
24
        def __init__(self, env_creator, COACHEnvClass, params, model_path):
25
26
             self.env_creator = env_creator
            self.COACHEnvClass = COACHEnvClass
27
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
             self.policy = PPO.load(model_path)
36
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
                 if all([a or b for a,b in zip(term.values(), trunc.values())]):
56
                     running = False
57
58
             coas = dict()
            for role, agent in self.env.coa_env.agents.items():
60
                 coas[role] = agent.coa
62
            traj = copy.deepcopy(self.env.coa_env.state.trajectory)
            return coas, traj
```

1.5 env.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
 2
 3
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
      You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
    # Unless required by applicable law or agreed to in writing, software # 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 svs
16
17
    from gymnasium.spaces import Box
18
    from pettingzoo.utils.env import ParallelEnv
19
    import gymnasium as gym
20
    from typing import Any
    import numpy as np import argparse
21
22
23
    from coach import (
24
25
        SeededParameterGenerator,
26
        BaseParameterGenerator.
27
28
29
    from utilities.planning import COA
30
    {\tt from\ coach\ import\ COACHEnvironment,\ CommunicationSchedule}
31
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
        the class to use in creating env objects _{\mbox{\tiny NNN}}
48
49
50
        return COACH_PettingZoo
51
52
    class COACH_PettingZoo(gym.Wrapper,ParallelEnv):
54
        def __init__(self, env_creator, COACHEnvClass):
             self.env_creator = env_creator
55
56
             self.COACHEnvClass = COACHEnvClass
57
             self.fake_render_mode = "rgb_array"
58
        def _setup(self, params):
59
60
             self.current_params = params
             self.COACH_params = COACH_params = params["COACH_params"]
             self.env_params = env_params = params["env_params"]
62
63
64
             # Set up parameter generator
             if COACH_params["stochastic"]:
65
66
                 self.parameter_generator = SeededParameterGenerator(23423, env_params)
67
             else:
68
                 self.parameter_generator = BaseParameterGenerator(23423, env_params)
69
70
             # Set up comm schedule
             if "FIXED_STEPS_PER_COM" in COACH_params.keys():
71
                 schedule_param = COACH_params["FIXED_STEPS_PER_COM"]
72
73
                 self.comm_schedule = CommunicationSchedule.repeating(**schedule_param)
74
             else:
                 logger.info("Communication Schedule Is Non-repeating")
75
76
                 self.comm_schedule = CommunicationSchedule(length=0)
77
             self.ACTION_PADDING = COACH_params.get("ACTION_PADDING", 0)
```

```
self.MIN_NEXT_ACTION_TIME = COACH_params.get("MIN_NEXT_ACTION_TIME", 1)
self.MAX_NEXT_ACTION_TIME = COACH_params.get("MAX_NEXT_ACTION_TIME", np.inf)
 79
 80
 81
 82
              # Setup Agents
 83
              agent_dict = dict()
 84
 85
              if "Agents" in self.COACH params.kevs():
 86
                  for role, agent in self.COACH_params["Agents"].items():
                      logger.info(f"########### {role} {agent}")
agent_class = self.COACHEnvClass.agent_selection.Agents[agent["class_name"]]
 87
 88
 89
                       agent_dict[role] = agent_class(role, **agent.get("params", dict()))
 90
 91
              # Create COA Env
              self.coa_env = self.COACHEnvClass(
 92
 93
                  env_creator=self.env_creator,
 94
                  parameter_generator=self.parameter_generator,
 95
                  agents=agent_dict,
 96
                  fill_random=False,
 97
                  {\tt 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_indexs = []
              idx = 0
117
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_indexs
124
                  for label, action in actions.items():
                       # Need to add an inital entry for the start time
                       lows.append(np.concatenate([[self.MIN_NEXT_ACTION_TIME],
126
                           np.array(action.low).reshape(-1)]))
                       highs.append(np.concatenate([[self.MAX_NEXT_ACTION_TIME],
127
                           np.array(action.high).reshape(-1)]))
128
129
                       L = len(np.array(action.low).reshape(-1)) + 1
130
                       self.action_indexs.append((role, label, idx, idx + L))
131
                       idx = idx + L
132
133
                  lows = np.concatenate(lows)
                  highs = np.concatenate(highs)
134
135
                  all_lows.append(lows)
136
                  all_highs.append(highs)
                  self.player_actions[role] = Box(low = lows, high = highs)
137
138
              self.action_spaces = {"director": Box(
139
140
                       low = np.concatenate(all_lows),
                       high = np.concatenate(all_highs)
141
                  )
142
              }
143
144
145
              # Observation spaces
146
              self.player_observations = dict()
147
              all lows = []
148
              all_highs = []
149
              for role in self.players: # Preserve order. Dicts should do this now but just to be safe
150
                  # Process actions
151
                  observations = self.coa_env.observation_spaces[role]
152
                  low = np.array(observations.low).reshape(-1)
153
                  high = np.array(observations.high).reshape(-1)
154
155
                  all_lows.append(low)
156
                  all_highs.append(high)
157
                  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
        def action_space(self, role):
171
            return self.action_spaces[role]
172
173
174
        def reset(self, seed=0, options=None):
175
            self.coa_env.reset()
176
            self.steps = 0
177
            self.cummulative_rew = np.zeros(len(self.possible_agents))
178
179
            self.coas = dict()
            for role in self.players:
180
181
                self.coas[role] = COA()
182
183
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 tructions
208
            term = False
209
            trunc = False
210
211
            if step_end["term_or_trunc"]:
212
                if all(list(last_returns[2].values())):
213
                    \mbox{\tt\#} Unless everyone terminates, somebody must have trucated.
214
                   term = True
215
                else:
216
                   trunc = True
217
218
            # Process reward
219
            reward_til_now = sum(self.coa_env.state.cummulative_rews.values())
220
            step_reward = reward_til_now - self.cummulative_rew
            self.cummulative_rew = reward_til_now
221
222
223
            return (
                {"director": self._process_observations(last_returns)},
224
                {"director": step_reward}, ## Reward
225
                {"director": term}, ## Term
226
                {"director": trunc}, ## Trunc
{"director": {}}, ## Info
227
228
            )
229
230
    231
232
    # Converstion between actions and COAs
233
    234
235
        {\tt 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_indexs:
242
               self.coa_env.state.current_t, action[i0+1:i1])
243
               event = {
244
                   "start": np.floor(action[i0]) + self.coa env.state.current t.
                   "label": label,
245
246
                   "parameters": action[i0+1:i1],
                   "role": role.
247
248
               }
               coas[role].append(event)
249
250
           for role in self.players:
251
252
               self.coas[role].add_events_from_dict(coas[role])
253
254
        def _process_observations(self, last_returns):
255
             When working with a specific env you almost certanly want to change this
256
            # as there may be a ton of redudenent information in the combined observation
257
            # space
            obs = np.concatenate([last_returns[0][role].reshape(-1) for role in self.players])
258
259
            return obs
260
    261
262
    # Wrapper Functions
    263
264
        def __getattr__(self, name: str) -> Any:
    """Returns an attribute with ``name``, unless ``name`` starts with an underscore."""
265
266
267
268
            if name == "coa_env":
269
               if "coa_env" not in self.__dict__.keys():
                   self.__dict__["coa_env"] = None
270
271
               return self.__dict__["coa_env"]
272
273
            if name in self.__dict__:
274
               return self.__dict__[name]
275
276
            if name == "unwrapped":
277
               return self.coa_env.env
278
279
           if name == "parallel_env":
               return self
280
281
282
           return getattr(self.coa_env.env, name)
283
    284
285
    # DASH Viewer Functions
    286
287
288
        def set_fake_render_mode(self, fake_render_mode):
289
            self.fake_render_mode = fake_render_mode
290
291
    # %%
292
    if __name__ == "__main__":
        params = {
293
            "COACH_params":{
294
295
               "stochastic": True,
296
               # "FIXED_STEPS_PER_COM": {
297
               #
                     "checkin_frequency": 10
298
               # },
299
               "ACTION_PADDING": 0,
               "MIN_NEXT_ACTION_TIME": 1,
300
               "MAX_NEXT_ACTION_TIME": 10,
301
               "Agents": {
302
                   "pursuer_0": {
303
                       "class_name": "DefaultActor",
304
                      "params": {"max_action_len":6}
305
306
307
                   "pursuer_1": {
                       "class_name": "DefaultActor".
308
                      "params": {"max_action_len":5}
309
310
                  }.
311
               "seed": 453413,
312
313
           },
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
                    print("#"*20, "COA Gym Information", "#"*20)
print("Players:", env.players)
print("Observation Space:", env.observation_spaces["director"].shape)
print("Action Space:", env.action_spaces["director"].shape)
print("Sample action:", env.action_spaces["director"].sample())
323
324
325
326
327
328
                   for i in range(50):
    print("Turn", i)
    obs,rew,term,trunc,info = env.step({"director": env.action_spaces["director"].sample()})
    if all([a or b for a,b in zip(term.values(), trunc.values())]):
        print("Environment has terminated.")
        break
329
330
331
332
333
334
335
                     env.render(ao="actions")
336
           # %%
337
```

1.6 params.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
 2
 3
    # Licensed under the Apache License, Version 2.0 (the "License");
      you may not use this file except in compliance with the License.
      You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
    # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
    # 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
    def get_env_param_class(args: Any) -> Type:
17
18
          ""Returns the class to use, based on input arguments
19
20
        Parameters
21
22
        args: argparse.Namespace
            arguments that were passed to the 'main()' function
23
24
25
        Returns
26
27
        class
        the class to use in creating env parameter objects """
28
29
30
        return EnvParams
31
32
33
    """Defines base class for environment parameter classes."""
34
35
    from typing import Any, Optional
36
37
38
    class EnvParams:
39
         """Base class for environment params."""
40
41
         def __init__(self, args: Any, param_section_name: str = "env_params") -> None:
42
             self.args = args
43
             self._params = getattr(args, param_section_name, {})
44
45
         def __getitem__(self, key: str) -> Any:
46
               "Return value stored for key from the params dict."""
47
             return self._params[key]
48
        def __setitem__(self, key: str, value: Any) -> None:
    """Set the value for key in the params dict."""
49
50
51
             self._params[key] = value
52
53
         def get(self, key: str, default: Optional[Any] = None) -> Any:
54
              ""Return value for key from the params dict, or None if it doesn't exist."""
55
56
                 return self._params[key]
57
             except KeyError:
58
                 return default
        def get_mutated_params(self) -> "EnvParams":
60
              ""Return a mutated copy of the params""
62
             raise NotImplementedError(
                 f"get_mutated_params has not been implemented in {type(self)}"
63
64
65
66
        def checkpoint(self, folder: str) -> None:
67
             """Save a checkpoint in the given folder."""
68
             raise NotImplementedError(
69
                 f"checkpoint has not been implemented in {type(self)}"
70
71
72
        def reload(self, folder: str) -> None:
              ""Read a checkpoint from the given folder."""
73
74
             raise NotImplementedError(f"reload has not been implemented in {type(self)}")
```

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

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

```
79
      def main() -> None:
 80
           parser = argparse.ArgumentParser(usage='%s [-d DIR] [-i extension ...]\n' % __file__
 81
                + 'example: %s -d ./src -i foo.m -i makefile .c .d .py\n\n' % __file__,
description='Will search under DIR for all source files with the specified file extensions, and
 82
 83
           compile them into a LaTeX file.')

parser.add_argument('--dir', '-d', help='root directory under which to search', default='.')

parser.add_argument('--include', '-i', action='append', help="Explicitly include a file even if it

doesn't match the extension list", default=[])

parser.add_argument('--style', default='default', choices=styles.keys(), help='Changes syntax
 84
 85
 86
                highlighting, etc. )
           parser.add_argument('extension', nargs='+', help='Only files with these extensions will be included
 87
                (leading dot optional)')
           args = parser.parse_args()
 88
 89
           # Permit valid extensions to be input with or without the dot args.extension = [ a if ('.' == a[0]) else '.%s' % a for a in args.extension ]
 90
 91
 92
 93
           # Make relative to base path, escape underscores
 94
           def format_path(path: str) -> str:
 95
 96
                if path == args.dir: return '/
                assert (path[0:len(args.dir)+1] == args.dir + '/') or (path[0:len(args.dir)+1] == args.dir +
 97
                     (1//1)
                return re.sub('_', r'\_', path[len(args.dir)+1:])
 98
 99
100
           # Print single file
           def dumpsrc(dirpath: str, fname: str) -> str:
101
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
                     s = r'\lstinputlisting{%s}' % path
109
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)
                print(s.strip())
114
115
116
           print_header()
117
           dirs = {dirpath:fnames for dirpath, _, fnames in os.walk(args.dir)}
includes = { os.path.realpath(f):f for f in args.include }
118
119
           for dirpath in sorted(dirs):
120
121
                fnames = dirs[dirpath]
122
                src = sorted([f for f in fnames
                     if (os.path.splitext(f)[1] in args.extension) or (os.path.realpath(f) in includes)])
123
                if 0 == len(src): continue
124
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
                     f = os.path.realpath(f)
131
                     if f in includes:
132
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}')
                for _,f in includes.items():
    f = args.dir + '/' + os.path.relpath(f, args.dir)
138
139
140
                     print(dumpsrc(os.path.dirname(f), os.path.basename(f)))
141
           print(r'\end{document}')
142
143
144 main()
```

2 DASH

2.1 DASH/app.py

```
# 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.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
    # Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 9
10
11
1.2
    # 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
   import copy
23
    import json
    # Import Dash Things
  from dash import Dash, html, dcc, Input, Output, callback, State, ALL, no_update, ctx, MATCH
    import plotly.express as px
28 import plotly.graph_objects as go
30 # Import Custom Things
    from DASH.html_objects import app_layout, plan_menu, actions_display
  from DASH.dash_utilities import callback_tools as ct
33
    # Import Env Things:
35
    from DASH.coach_integration import COACHIntegration
36
   from coach import COACHEnvironment
37
38
    # Logging
39
    import logging
40
    logging.basicConfig(level=logging.DEBUG, format="%(levelname)s:%(name)s:%(message)s")
41
42
    logger = logging.getLogger(__name__)
43
    pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
44
         name.startswith("pymunk")]
45
46
    for log_handler in pymunk_loggers:
        log_handler.setLevel(logging.INFO)
47
48
    49
50
    # Inputs
    51
52
53
    ### Plan Selector
54
55
    @callback(
        Output("update_backend_coa", "data", allow_duplicate=True),
Input({"type":"plan", "id": ALL}, "n_clicks"),
56
57
        prevent_initial_call=True,
58
    )
59
60
61
    def select_plan(plan):
62
        logger.debug("INPUT: select_plan")
63
64
        plan_id = ctx.triggered_id['id']
        logger.debug("\t %s %s %s","select_plan", plan, plan_id) if plan[plan_id] > 0:
65
66
67
             ct.env_factory.plans.set_current_plan(plan_id)
68
69
        return True
70
71
    ### Plan View Checkboxes
    @callback(
```

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

```
152
                  logger.debug("\t %s %s", "calling id:", id)
153
154
                  if ctx.triggered_id is None:
                          ## This got called by a plan change
155
156
                          return no_update
157
                  logger.debug("\t %s %s", "ctx.triggered_id:", ctx.triggered_id)
158
159
                  role = ctx.triggered_id['role']
160
                  logger.debug("\tilde{it} \tilde{ /s} \tilde{ \tilde{'s}} \tilde{ \tilde{'s}} \tilde{ \tilde{ \tilde{ 'New Params}:}} \tilde{ \tilde{
161
162
163
                  locks["action_card"] = True
                  logger.debug("\t %s","LOCKING ACTION PARAMS")
164
165
166
                  i = id.index(ctx.triggered_id)
                  plan = ct.env_factory.plans.get(ctx.triggered_id["plan"])
167
168
                  coa = plan.coas[ctx.triggered_id["role"]]
                  event = coa.get(time = ctx.triggered_id["time"], label = ctx.triggered_id["type"])
169
170
171
                  logger.debug("\t %s %s","Param_Update: Current COA" , coa)
172
173
                  event.parameters[ctx.triggered_id["index"]] = params[i]
174
                  logger.debug("\t %s %s","Param_Update: New COA" , coa)
175
176
177
                  return True
178
179
         ### Timeline Selector
180
181
          @callback(
                  Output("update_backend_coa", "data", allow_duplicate=True),
Input({"type": "timeline", "role": ALL}, "value"),
182
183
184
                  prevent_initial_call=True,
185
         )
186
187
          def edit_timeline(new_timelines):
188
                  role = ctx.triggered_id['role']
189
                  logger.debug("\t %s","INPUT: edit_timeline")
logger.debug("\t %s %s","new_timelines", new_timelines)
logger.debug("\t %s %s","Old COA:", ct.env_factory.plans.current.coas[role])
190
191
192
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
                 for j_old, j_new in zip(old_timeline, changed_timeline):
    if not j_old == j_new:
209
210
                                  old.append(j_old)
211
212
                                  new.append(j_new)
213
                  if len(old) == 0:
214
                          logger.debug("INPUT: edit_timeline - no_update")
215
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]
                                  j_old = old[0]
222
                          else:
223
224
                                 j_new = new[0]
                                 j_old = old[1]
225
226
                  else:
                         j_new = new[0]
j_old = old[0]
227
228
229
230
                          # Check if we're moving one point on top of another
                          if j_new in old_timeline:
231
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)
                while j_new in changed_timeline:
234
                     j_new += sign*1
235
236
         237
238
239
240
241
         ct.env_factory.plans.current.coas[role].move_events(time=j_old, to=j_new)
242
         logger.debug("\t %s %s","New COA:", ct.env_factory.plans.current.coas[role])
243
244
245
         logger.debug("INPUT: edit_timeline - Update Backend")
246
         return True
247
    248
249
    # Ruttons
250
    251
252
    ### New Plan
253
    @callback(
         Output("update_backend_coa", "data", allow_duplicate=True),
Input("button_new_plan", "n_clicks"),
254
255
256
         prevent_initial_call=True,
257
258
     def new_plan(nclicks):
         logger.debug("BUTTON: new_plan")
259
260
         ct.env_factory.plans.new_plan()
261
262
         return True
263
264
    ### New Command
265
     @callback(
         Output("update_backend_coa", "data", allow_duplicate=True),
266
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(
         Output("update_backend_coa", "data", allow_duplicate=True), Input("button_simulate_plan", "n_clicks"),
282
283
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
         ct.env_factory.run_plan(current_plan)
289
290
291
        return True
292
293
294
    ### Generate Plan
295
296
    Ocallback(
        Output("update_backend_coa", "data", allow_duplicate=True),
Input("button_generate_plan", "n_clicks"),
297
298
299
         prevent_initial_call=True,
300
301
     def simulate_plan(nclicks):
        logger.debug("BUTTON: generate_plan")
302
303
         ct.env_factory.generate_plan()
         return True
304
305
306
    307
308
    # Outputs
309
    310
311
     @callback(
         Output({"type": "onboard_model", "role": ALL}, "options", allow_duplicate=True),
Output({"type": "onboard_model", "role": ALL}, "value", allow_duplicate=True),
312
313
```

```
Output({"type": "timeline", "role": ALL}, "value", allow_duplicate=True),
Output({"type": "agent_interface", "role": ALL}, "value", allow_duplicate=True),
Output({"type": "actions_card", "role": ALL}, "children", allow_duplicate=True),
Output("table_plans", "children", allow_duplicate=True),
314
315
316
317
           Output('button_generate_plan', 'disabled'),
318
           Input("update_frontend_elements", "data"),
319
320
           prevent initial call=True.
     )
321
322
      def display_choose_model(update):
323
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())
           interface = ct.make_returns_from_dict(ct.env_factory.plans.current.interfaces)
328
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
           logger.debug("\t %s %s","Models", possible_models)
logger.debug("\t %s %s","Selected Model", selected_model)
logger.debug("\t %s %s","Timelines", timelines)
logger.debug("\t %s %s","Interfaces", interface)
335
336
337
338
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_visialization", "figure", allow_duplicate=True),
347
           Input('update_frontend_elements', 'data'),
348
           prevent_initial_call=True,
349
350
351
      def visualization(update_frontend_elements):
           logger.debug("OUTPUT: visualization")
352
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
           Output("stats-graphic-1", "figure", allow_duplicate=True),
Output("stats-graphic-2", "figure", allow_duplicate=True),
Output("stats-graphic-3", "figure", allow_duplicate=True),
Input('update_frontend_elements', 'data'),
363
364
365
366
367
           prevent_initial_call=True,
     )
368
369
370
      def telemetry(update_frontend_elements):
           logger.debug("OUTPUT: telemetry")
371
372
           figures = []
373
           for plan_id in ct.env_factory.plans.active:
374
                logger.debug("\t %s %s", "active plan_id", plan_id)
375
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:</pre>
                               logger.debug("\t %s %s %s", "adding figure", len(figures), i)
381
382
                               f = go.Figure()
383
                               f.update_layout(
                                    margin={"l": 5, "b": 0, "t": 0, "r": 80},
384
                                    hovermode="closest",
385
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
                         one.")
396
                      opacity = 1
397
398
                  for col in t.columns:
                      figures[i].add_trace(go.Scatter(x=t.index,y=t[col],
399
400
                         mode='lines',
name=f"{plan_id}_{col}",
401
402
                          opacity=opacity
403
404
405
        if len(figures) == 0:
406
           return no_update
407
408
        while len(figures) <3:
           figures.append(go.Figure())
409
410
        return figures
411
412
    413
    # Update Backend
414
    415
416
417
    @callback(
        Output("update_frontend_elements", "data", allow_duplicate=True),
418
        Input("update_backend_coa", "data"),
419
420
        # prevent_initial_call=True,
421
        prevent_initial_call='initial_duplicate'
422
    )
423
    def update_coa(coa_update):
424
        logger.debug("BACKEND: update_coa")
425
426
        return True
427
428
    429
    # Setup
430
    431
432
433
               "action_card": False
434
435
    if __name__ == "__main__":
    logger.debug("app - ################ RELOADING DASH APP ###################")
436
437
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
       app = Dash(__name__)
app.layout = app_layout(env_factory)
443
444
445
       app.run(debug=True)
```

2.2 DASH/coach_integration.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
3
    # Licensed under the Apache License, Version 2.0 (the "License");
4
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
    \mbox{\#} Unless required by applicable law or agreed to in writing, software \mbox{\#} distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # 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
   import json
24
    import os
    import copy
    import numpy as np
    from itertools import product
    import logging
29 logger = logging.getLogger(__name__)
31
    from coach import (
        SeededParameterGenerator,
33
        BaseParameterGenerator,
34
    )
35
36
    import agents as agents_module
    import directors as directors_module
37
38
39
40
    def role type(role):
        return role.split("_")[0]
41
42
    class COACHIntegration:
43
44
        def __init__(self,
45
             env creator.
46
             COACHEnvClass
47
            parameters,
48
             agents_module=agents_module
        ) -> None:
49
50
            self.agents_module=agents_module
51
            self.env_creator = env_creator
             self.COACHEnvClass = COACHEnvClass
52
            self.parameters = parameters
self.render_gif = False
53
54
55
            self.library = dict()
56
            self._id_to_interface = dict()
57
            self._interface_to_id = dict()
58
59
60
            self.setup()
61
62
             self.plans = PlanLibrary(
63
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
                 self.plans.current.interfaces.items()}
72
73
        # def get_interfaces(self):
74
               return list(self.interfaces.keys())
76
        def set_current_interface(self, role, interface):
            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
         def get_current_model(self):
 88
 89
              return self.current model
 90
         def set_current_model(self, role, model):
 91
 92
              self.current_model[role] = model
 93
 94
         def actions(self, role):
 95
              agt = self.coach_env.agents[role]
              return agt.interface.action_dictionary
 96
 97
         # Deal with Plan Library
 98
         def new_plan(self, name = None):
99
100
              self.plans.new_plan(name)
101
102
          def get(self, id):
103
              return(self.library[id])
104
105
          def kevs(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
              \mbox{\tt \#\#} COA_Env has agents, need to extract them from env.
134
             # Create COA Env
135
              self.COACH_params = COACH_params = self.parameters["COACH_params"]
136
137
              self.env_params = env_params = self.parameters["env_params"]
138
              self.actor_params = actor_params = self.parameters["actor_params"]
139
              # Set up parameter generator
140
              if COACH_params["stochastic"]:
141
                  self.parameter_generator = SeededParameterGenerator(23423, env_params)
142
143
              else:
                  self.parameter_generator = BaseParameterGenerator(23423, env_params)
144
145
146
              # Set up comm schedule
              if "FIXED_STEPS_PER_COM" in COACH_params.keys():
schedule_param = COACH_params["FIXED_STEPS_PER_COM"]
147
148
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)
              self.MIN_NEXT_ACTION_TIME = COACH_params.get("MIN_NEXT_ACTION_TIME", 1)
self.MAX_NEXT_ACTION_TIME = COACH_params.get("MAX_NEXT_ACTION_TIME", np.inf)
155
156
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
              self.role_to_idx = {role:idx for idx, role in enumerate(self.roles)}
self.idx_to_role = {idx:role for idx, role in enumerate(self.roles)}
171
172
173
              self.role_types = list(set([role_type(role) for role in self.roles]))
174
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()
              self.model_classes = dict()
184
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
                        \verb|self.coach_env.\__class\__.agent\_selection.Interfaces[if\_params["interface\_class"]]|
193
194
                   # Get a reference interface, this is make sure that any model class instituted has the
195
                   role = if_params.get("roles", self.roles)[0] # Get one applicable role
196
                   if_reference = if_Class(role, self.coach_env.env, **if_params.get("iterface_parameters",
                        dict()))
197
                   logger.debug("Current Class: %s", if_reference)
198
199
                   # Set up references to interfaces
200
                   self.interfaces[if_name] = if_reference
201
202
                   # Get roles for which the interface is valid
                   for role in if_params.get("roles", self.roles):
203
                       self.interfaces_by_role[role][if_name] = if_reference
204
205
                       if if_name not in self.roles_by_interface.keys():
206
                            self.roles_by_interface[if_name] = []
207
                       self.roles_by_interface[if_name].append(role)
208
209
                   # Setup reference to model creation information
210
                   self.model_params_by_interface[if_name] = if_params["models"]
211
212
                   self.model_classes[if_name] = dict()
                  for model_name, model_params in if_params["models"].items():
    self.model_to_interface[model_name] = if_name
213
214
215
                       self.model_classes[if_name][model_name] =
                       self.agents_module.__dict__.get(model_params["class_name"])
self.model_params[model_name] = model_params
216
217
218
              # Setup Directors
219
              directors = self.actor_params["directors"]
              self.directors = dict()
220
221
              self.directors_allow = dict()
222
223
              for dr_name, dr_params in directors.items():
224
                   {\tt self.default\_models} \ = \ \{ {\tt role: params["classes"][0]} \ \ {\tt for role, params in} \\
225
                        dr_params['roles'].items()}
226
                   self.default_interfaces = {role: self.model_to_interface[model] for role, model in
                        self.default_models.items()}
227
228
                   tmp_params = copy.copy(self.parameters)
229
                  for role, params in dr_params["roles"].items():
    tmp_params["COACH_params"]["Agents"][role] = self.model_params[params["classes"][0]]
230
231
232
233
                   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
                  class_iter = product(*[classes["classes"] for classes in dr_params["roles"].values()])
241
242
                  for t in class_iter:
243
                      self.directors_allow[tuple(t)] = self.directors[dr_name]
244
245
                      self.model to interface
246
         def generator_available(self):
247
             return tuple(self.plans.current.models.values()) in self.directors_allow.keys()
248
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
             tmp.trajectory = traj
258
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):
             logger.debug("\t plan info: %s %s %s", plan.name, plan.interfaces, plan.models)
267
268
269
270
             if self.render_gif:
271
                  traj = self.coach_env.get_traj_from_plan(plan, render=True)
272
                  frames = np.array(traj.frames)
                 plan.visualizations = px.imshow(frames, animation_frame=0, binary_string=True, labels=dict(animation_frame="slice"))
273
274
275
                 traj = self.coach_env.get_traj_from_plan(plan, render=False)
276
277
                  plan.visualizations = self.coach_env.plot_trajectory_component(
278
279
                      plan=plan,
280
                      env=self.coach_env.env,
281
                      render_mode="plotly"
282
283
284
             plan.trajectory = traj
285
             plan.telemetry = self.coach_env.trajectory_telemetry(traj, plan, self.coach_env.env)
286
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:
                 plan.name = name
295
296
297
             plan.coas = coas
298
             return plan
299
300
         def __init__(self,
             roles,
name = None,
301
302
303
             interfaces_by_role=None,
304
             model_classes_library=None,
305
             model_params_library=None
306
             ):
             self.id = copy.copy(Plan.id)
307
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
              self.coas = {role: COA() for role in self.roles}
self.interfaces = {role: None for role in self.roles}
319
320
              self.models = {role: None for role in self.roles}
321
              self.model_classes = {role: None for role in self.roles}
self.model_params = {role: dict() for role in self.roles}
322
323
              self.trajectories = {role: None for role in self.roles}
324
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):
              if hasattr(interface, "__len__"):
    if len(interface) == 0:
342
343
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",
                   dict())
359
360
          def get_model(self, role):
361
              return self.models[role]
362
363
          def get_timelines(self):
364
              return {role: self.coas[role].to_dict() for role in self.roles}
365
366
          def get_dash_timelines(self):
367
              timelines = self.get_timelines()
368
              for k,v in timelines.items():
369
                  if len(v) == 0:
370
                       timelines[k] = []
371
                  else:
372
                      timelines[k] = list(v.keys())
373
              return timelines
374
375
         def new event(self, role):
376
              logger.debug("Adding new event to COA: %s", self.coas[role])
377
              action_name, act_box = list(self.get_interface(role)[1].action_dictionary.items())[0]
378
379
380
              ## [{"start":int, "action": str, "parameters":np.array}]
381
              last_t = -1
              if len(list(self.coas[role].timeline.keys()))>0:
382
383
                  last_t = list(self.coas[role].timeline.keys())[-1]
384
385
              logger.debug("Action Name: %s", action_name)
386
              tmp = TimelineEvent(
387
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,
              plans = [],
401
402
              default_interfaces=None,
403
              default models=None
404
             ):
             self.roles = env_factory.roles
405
406
              self._plans: list[Plan] = plans
407
              self.current: Plan = None
408
              self.active = set()
              self.interfaces_by_role = env_factory.interfaces_by_role
409
             self.model_classes_library = env_factory.model_classes
self.model_params_library = env_factory.model_params_by_interface
410
411
412
413
              if not default_interfaces:
414
                  self.default_interfaces = {role: env_factory.get_interfaces_by_role(role)[0] for role in
                       self.roles}
415
             else:
416
                  self.default_interfaces = default_interfaces
417
418
              if not default_models:
419
                  self.default_models = {role:
                       env_factory.get_interface_models(self.default_interfaces[role])[0] for role in
                       self.roles}
420
421
                  self.default_models = default_models
422
423
              self.new_plan()
424
              self.current = self.all()[0]
425
              self.active.add(self.current.id)
426
427
428
              return list(self._id_to_plan.keys())
429
430
         def get(self, id):
431
              return self._id_to_plan[id]
432
433
         def set_current_plan(self, id):
434
              self.current = self._id_to_plan[id]
435
436
         def new_plan(self, name=None):
              tmp = Plan(
437
438
                  self.roles,
439
                  name=name,
440
                  interfaces_by_role = self.interfaces_by_role,
441
                  model_classes_library=self.model_classes_library,
                  model_params_library=self.model_params_library
442
443
444
445
             for role, interface in self.default_interfaces.items():
                  tmp.set_interface(role, interface)
tmp.set_model(role, self.default_models[role])
446
447
448
449
              self.add_plan(tmp)
450
             return tmp
451
452
         def add_plan(self, plan: Plan):
              self._plans.append(plan)
453
              self._id_to_plan = {plan.id:plan for plan in self._plans}
454
455
456
         def all(self):
              return self._plans
457
458
459
460
     class COALibrary:
461
         def __init__(self, roles, coas: dict[str, COA] = None):
462
463
              self.roles = roles
              self.role_types = list(set([role_type(role) for role in roles]))
464
465
466
              self._library = {role_type:[] for role_type in self.role_types}
467
468
              if coas is not None:
469
                  for role_tag, coa in coas.items:
470
                      self.add_coa(role_tag, coa)
471
```

```
472
            def add_coa(self, role_tag, coa):
    if role_tag in self.roles:
        role_type = role_tag
    elif role_type(role_tag) in self.roles:
        role_type = role_type(role_tag)
473
474
475
476
477
478
                  else:
                       479
480
                  if coa not in self._library[role_type]:
    self._library[role_type].append(coa)
481
482
483
            def __setitem__(self, key, item):
    self._library[key] = item
484
485
486
            def __getitem__(self, key):
    return self._library[key]
487
488
489
            def values(self):
490
                  return self._library.values()
491
492
            def items(self):
    return self._library.items()
493
494
495
             def keys(self):
496
497
                  return self._library.keys()
```

2.3 DASH/dash_utilities.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
3
    # Licensed under the Apache License, Version 2.0 (the "License");
4
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
    \mbox{\#} Unless required by applicable law or agreed to in writing, software \mbox{\#} distributed under the License is distributed on an "AS IS" BASIS,
10
    # 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)
            i = callback_tools.env_factory.roles.index(role)
            returns[i] = r
            return(returns)
31
        @staticmethod
        def get_role_from_callback(role, callback):
             # 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():
                     returns[i] = r[role]
41
42
            return callback_tools.nones_to_empty_list(returns)
43
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:
                 return {k:callback_tools._ntol(v) for k, v in r.items()}
54
55
56
        @staticmethod
57
        def _ntol(r):
58
            if r is None:
59
                 return []
60
61
            return r
```

2.4 DASH/html_objects.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
    # You may obtain a copy of the License at
6
          http://www.apache.org/licenses/LICENSE-2.0
8
    \mbox{\tt\#} Unless required by applicable law or agreed to in writing, software \mbox{\tt\#} distributed under the License is distributed on an "AS IS" BASIS,
Q
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
11
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
         for role in env_factory.roles:
21
22
             RoleInterfaceSelect.append(html.Div(
23
                      Γ
24
                           html.Div([
25
                               role,
26
                                dcc.Dropdown(
                                    env_factory.get_interfaces_by_role(role),
28
                                    placeholder=f"Interface",
                                    id={"type": "agent_interface", "role": role}
30
31
                                dcc.Dropdown(
                                    id={"type": "onboard_model", "role": role},
                                    placeholder=f"Model"
33
34
35
                                html.Button("Add New Command", id={"type": "button_add_new_command", "role":
                                    role}),
36
                               className="left_card",
style={"float": "left", "margin": "auto"},
37
38
39
40
                           html.Div(
                               className="right_card",
id={"type": "actions_card", "role": role},
style={"float": "left", "margin": "auto"},
41
42
43
44
45
                       1.
46
                       className="row agent_card",
47
48
         RoleInterfaceSelect = html.Div(RoleInterfaceSelect, className="action_cards")
49
50
         ## The Role Timeline Selection
TimeLines = []
51
52
53
         for role in env_factory.roles:
54
             TimeLines += [
                  html.Tr([
55
                       html.Td(role, style={"float": "left", "margin": "auto", "width": 100}),
56
57
                       html.Td(
58
                           dcc.RangeSlider(
                               Ο,
59
60
                               100.
61
                                value=[]
                                tooltip={"placement": "bottom", "always_visible": True},
62
                               id={"type": "timeline", "role": role},
63
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)
         ## App Layout
76
         app_layout = html.Div(
```

```
78
                     html.Div(
 79
                          Γ
 80
                               html.Div(
 81
                                    Γ
 82
                                         dcc.Graph(
                                              id="plotly_visialization",
style={"width": "600px", "height": "600px"},
 83
 84
 85
 86
                                    ٦.
                                    style={"width": "48%", "display": "inline-block"},
 87
 88
                               ).
 89
                               html.Div(
 90
                                    Ε
                                         dcc.Graph(id="stats-graphic-1", style={"height": "150px"}),
dcc.Graph(id="stats-graphic-2", style={"height": "150px"}),
dcc.Graph(id="stats-graphic-3", style={"height": "150px"}),
 91
 92
 93
 94
                                    style={"width": "48%", "float": "right", "display": "inline-block"},
 95
 96
                               ).
                          ]
 97
 98
                     ).
                     TimeLines,
 99
                    html.Button("New Plan", id="button_new_plan"),
html.Button("Generate Plan", id="button_generate_plan"),
html.Button("Simulate Plan", id="button_simulate_plan"),
100
101
102
103
                     ## These are True False values, they signle that the corresponding object should be updated.
104
                     dcc.Store(id="update_backend_coa"),
                     dcc.Store(id="update_frontend_elements"),
105
106
                     ### Agent Card:
107
                     html.Div(RoleInterfaceSelect),
108
                     html.Div([
109
                          "Plans"
                          html.Table(id="table_plans")
110
                     ], id="div_plans", style = {"width":"15%", "float": "right", "display": "inline-block",
111
                           "padding":"10px", "border":"1px solid black"})
112
                ],
113
           )
114
115
           return app_layout
116
117
      def action_table_row(time, event, interface, plan="", role=""):
118
           CELL_WIDTH = "80px"
119
120
121
           ## Action Label:
122
           html_objects = []
123
124
           ## Create the header describing the actions
           action_head = [
125
                html.Td("", style={"padding-left": "25px", "width": "200px"}),
html.Td("Time", style={"width": CELL_WIDTH, 'font-size': 10, 'font-style': 'italics'}),
126
127
128
129
130
           action_params = interface.action_dictionary[event["label"]]
131
132
           for param in action_params.items():
                # 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 = \Gamma
                html.Td(
140
141
                    Γ
142
                          html.Div(
                               className=f"rc-slider-handle rc-slider-handle-{time}",
style={"margin-top": "0px", "margin-right": "5px"},
143
144
145
                          html.Span(event["label"], style={"padding-left": "25px"}),
146
147
                     style={"width": "200px"},
148
                )
149
           ٦
150
151
152
           TDs.append(
153
                html.Td(
154
                     dcc.Input(
155
                               "type": "actionstart",
156
```

```
"plan": plan,
157
                           "time": time,
158
                           "type": event['label'],
159
                           "role": role
160
161
                       }.
                       type="number",
162
163
                       value=time.
                       placeholder=f"Step",
164
                       style={"width": CELL_WIDTH},
165
                       readOnly=True,
166
167
                       debounce = True,
168
                  )
169
              )
         )
170
171
          # print("Event Parameters:", event["parameters"])
172
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={
                                    "type": "actionparam",
180
                                    "plan": plan,
181
                                    "time": time;
182
                                    "type": event['label'],
"index": i,
183
184
185
                                    "role": role
186
                               },
                                type="number"
187
188
                                min=param.low[j]
189
                                max=param.high[j],
                               value=event["parameters"][i],
placeholder=f"[{param.low[j]:.3}, {param.high[j]:.3}]", # the :.3 is the number
190
191
                                    of significant figures
192
                                style={"width": CELL_WIDTH},
193
                               debounce = True,
194
                           )
195
                       )
196
197
198
          # print("Event Parameters:", TDs)
199
          html_objects.append(html.Tr(TDs))
200
201
          return html_objects
202
203
     def actions_display(env_factory):
204
          current_plan = env_factory.plans.current
html_objects = []
205
206
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()):
                       html_sub_objects += action_table_row(time, event, interface, plan=current_plan.id,
218
                            role=f"{role}")
219
220
              html_objects.append(html.Div(html_sub_objects))
221
222
          return html_objects
223
224
225
     def plan_menu(env_factory):
226
          html_objects = []
227
          for plan in env_factory.plans.all():
              if plan == env_factory.plans.current:
    display_class = "current_plan"
228
229
230
                  style = {"background-color":"#ccc", "border":0, "margin":0, "padding": "10px 40px 10px 40px"}
231
              else:
232
                  display_class = "archived_plan"
                  style = {"background-color": "#eee", "border":0, "margin":0, "padding": "10px 40px 10px 40px"}
233
234
235
              print("\t","plan_menu: plan", plan.name, display_class)
```

3 DASH/Documentation

3.1 DASH/Documentation/interface.txt

```
### Interface Selector
 3
     @callback(
           Output("onboard_model", "options", allow_duplicate=True),
           Output("model_table_body", "children", allow_duplicate=True),
Output("model_table_body", "value", allow_duplicate=True),
Input("agent_interface", "value"),
           prevent_initial_call=True,
     def choose_interface(value):
                return [o.__name__ for o in idx], Tb_children, []
12
13
     ### Model Selector
     @callback(
           Output("model_table_head", "children", allow_duplicate=True),
15
           Output ("model_table_subhead", "children"),
16
           Output("coa_update", "data", allow_duplicate=True), Input("onboard_model", "value"),
17
18
19
           prevent_initial_call=True,
20
21
     def select model(value):
           return table_head, table_subhead, True
23
24
25
     ### Add Action Button
26
     @app.callback(
          p.callback(
Output("my-range-slider", "value"),
Output("model_table_body", "children"),
Input("AddNewCommand", "n_clicks"),
State("my-range-slider", "value"),
State("model_table_body", "children"),
State("agent_interface", "value"),
27
28
29
30
31
32
33
     def add_COA_command(val, slider_values, children, interface):
34
35
                return slider_values, children
36
37
     ### Slider Action
38
39
     @callback(
           Output({"type": "actionstart", "index": ALL}, "value"),
40
           Output("coa_update", "data", allow_duplicate=True),
Input("my-range-slider", "drag_value"),
State("agent_interface", "value"),
41
42
43
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(
           Output("coa_update", "data", allow_duplicate=True),
Input({"type": "actionparam", "index": ALL}, "value"),
52
53
54
           State("agent_interface", "value"),
55
           prevent_initial_call=True,
56
     def update_coa_parameters(value, interface):
58
           return interface
60
     ### Process COA Update
     @callback(
           Output("indicator-graphic", "figure"),
Output("stats-graphic-vel", "figure"),
Output("stats-graphic-fuel", "figure"),
Output("stats-graphic-acc", "figure"),
63
64
65
66
           Input("coa_update", "data"),
67
68
           State("agent_interface", "value"),
69
     def process_coa_update(value, interface):
70
71
           return fig, fig1, fig2, fig3
72
73
     ### Generate COA
     @callback(
```

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

3.2 DASH/Documentation/network_view.py

```
# %%
1
    import networkx as nx
3
    import matplotlib.pyplot as plt
 4
5
    with open("../app.py", "r") as f:
   in_callback = False
   callbacks = []
 6
8
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"):
^{22}
                    in_callback = True
24
    # %%
    info = dict()
30
    for d, c in callbacks:
31
          Outputs = []
          Input = []
32
          State = []
33
34
35
          c = c.strip()
          cs = c.split(",")
36
         i = 0
37
38
          while i < len(cs):
              st = cs[i].strip()
39
40
              if st.startswith("Output"):
41
                   target = st.split("(")[1].replace("'","").replace('"',"")
atr = cs[i+1].replace("'","").replace(""',"").replace(")","")
Outputs.append([target,atr])
42
43
44
45
46
              if st.startswith("Input"):
47
                   target = st.split("(")[1].replace("'","").replace('"',"")
atr = cs[i+1].replace("","").replace(""","").replace(""","")
48
49
                    Input.append([target,atr])
50
51
                    i +=1
52
              if st.startswith("State"):
53
                   target = st.split("(")[1].replace("'","").replace('"","")
54
                   atr = cs[i+1].replace("'","").replace('"',"").replace(")","")
State.append([target,atr])
55
56
57
                   i += 1
58
              i+=1
59
60
61
          name = d.split()[1].split("(")[0]
62
          info[name] = {
                              "Input": Input,
"Output": Outputs,
63
64
                              "State": State
65
66
                         }
67
68
    info
69
    # %%
70
    edges = []
     functions = []
    inputs = []
73
    outputs = []
    states = []
    for name, s in info.items():
         functions.append(name)
         for i in s["Input"]:
```

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

4 DASH/assets

4.1 DASH/assets/typography.css

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

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

```
157
    .rc-slider-handle-4:hover {
158
159
        border-color: #ff006d;
160 }
161
162 /* ========= */
163
165 border-color: #adff02; 166 }
164 .rc-slider-handle-5 {
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
    .rc-slider-handle-6.rc-slider-handle-click-focused:focus {
183 border-color: #8f00ff; 184 }
182
185
187 border-color: #8f00ff; 188 }
    .rc-slider-handle-6:hover {
189
190 /* ========= */
```

5 examples/MAInspection

5.1 examples/MAInspection/Readme.md

```
1  # Multiagent Inspection
2
3  This example shows how to solve a simple satelite 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  <img src="Interface.png">
```

5.2 examples/MAInspection/app.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
    # You may obtain a copy of the License at
6
         http://www.apache.org/licenses/LICENSE-2.0
    \mbox{\tt\#} Unless required by applicable law or agreed to in writing, software \mbox{\tt\#} distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
11
12
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
15
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
    from examples.MAInspection.Environments.MAIcoach import MAI_COACH
25 import examples.MAInspection.Environments.MAIagents as MAIagents
26
    # Logging
    import logging
    logging.basicConfig(level=logging.DEBUG, format="%(levelname)s:%(name)s:%(message)s")
    logger = logging.getLogger(__name__)
31
    pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
        name.startswith("pymunk")]
33
    numba_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
         name.startswith("numba")]
34
    for log_handler in pymunk_loggers + numba_loggers:
        log_handler.setLevel(logging.INFO)
35
36
37
38
    def get_env():
        return PettingZooEnv(PZGame=Multinspect)
39
40
    if __name__ == "__main__":
41
        -- -- print("app - ################## RELOADING DASH APP #######################")
42
43
44
        with open(os.path.join("Experiment", "train_director.yaml"), "r") as f:
45
                 params = yaml.safe_load(f)
46
             except yaml.YAMLError as exc:
47
48
                 print(exc)
49
50
        with open(os.path.join("Experiment", "dash_app.yaml"), "r") as f:
51
52
                 params["actor_params"] = yaml.safe_load(f)
53
             except yaml.YAMLError as exc:
54
55
                 print(exc)
56
        # external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
57
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
        proxy_url = re.sub("{{port}}", "8050", os.environ["VSCODE_PROXY_URI"])
proxy_url = re.sub(os.environ["ACEHUB_BASEURL"], "", proxy_url)
73
         app = Dash(serve_locally=True, requests_pathname_prefix=proxy_url)
75
        app.layout = app_layout(env_factory)
```

78 app.run(debug=True)

5.3 examples/MAInspection/train_agents.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
2
    \mbox{\tt\#} Licensed under the Apache License, Version 2.0 (the "License");
3
4
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
   # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
   from __future__ import annotations
    import sys
16
    sys.path.insert(0, "../../")
17
18
19
   import examples.MAInspection.Environments.env as TrainingEnv
   import supersuit as ss
20
21 from stable_baselines3 import PPO
   from stable_baselines3.ppo import MlpPolicy
23 from stable_baselines3.common.callbacks import EvalCallback
    from stable_baselines3.common.vec_env import VecVideoRecorder
25 \quad \hbox{from PIL import Image} \\
    import os
    import time
    import glob
   from tqdm import tqdm
   import numpy as np
   import yaml
   import copy
33
34
    import torch
35
    CORES = 7
36
    torch.set_num_threads(CORES)
    torch.set_num_interop_threads(CORES)
38
39 # Code adapted from https://pettingzoo.farama.org/tutorials/sb3/waterworld/
40
    ## We're going to train two policies, one that assumes a densely poisonous env
41
42
    ## and one that requires sparse exploration.
43
44
    def train_butterfly_supersuit(
45
        env_fn,
        steps: int = 10_000,
46
        seed: int | None = 0,
47
48
        num_vec_envs=1,
        num_cpus=CORES.
49
50
        learning_rate=1e-3,
51
        batch_size=256,
52
        model_dir="",
name="",
53
        **env_kwargs
54
   ):
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,
             base_class="stable_baselines3")
67
68
        # Note: Waterworld's observation space is discrete (242,) so we use an MLP policy rather than CNN
69
        model = PPO(
70
            MlpPolicy,
71
72
             verbose=3,
             learning_rate=learning_rate,
             batch_size=batch_size,
             tensorboard_log=os.path.join("./tensorboard_log/", name)
76
```

```
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)
          eval_env = ss.concat_vec_envs_v1(eval_env, num_vec_envs=1, num_cpus=num_cpus,
 81
              base_class="stable_baselines3")
 82
          eval_callback = EvalCallback(eval_env, verbose=1, eval_freq=1000)
 83
 84
          \label{lem:weight_name} weight\_name \ = \ f"\{env.unwrapped.metadata.get('name')\}_{\{time.strftime('%Y%m%d-%H%M%S')\}"}
 85
          weight_path = os.path.join(model_path, weight_name)
 86
 87
          os.makedirs(weight_path,exist_ok=True)
 88
 89
          model_args = {
              "name": name,
"load_class": "SB_PPOWaypointerActor",
"env_params": env_kwargs
 90
 91
 92
 93
          7
 94
          with open(os.path.join(weight_path, "params.yaml"), "w") as f:
    yaml.dump(model_args, f, default_flow_style=False)
 95
96
 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,
          num_games: int = 5,
112
113
          render_mode: str | None = "rgb_array",
114
          render = True,
115
          **env_kwargs
116
     ):
117
          # Evaluate a trained agent vs a random agent
          env = env_fn(render_mode=render_mode, **env_kwargs)
118
119
120
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 = PPO.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 he same model for
              every agent
132
          for i in tqdm(range(num_games)):
133
              obs, infos = env.reset(seed=i)
              cumm_rewards = {agent: 0 for agent in env.possible_agents}
134
              running = True
frames = []
135
136
137
138
              while running:
139
                  acts = dict()
                  for role in env.agents:
140
                       acts[role] = model.predict(obs[role], deterministic=True)[0]
141
142
                  obs, rewards, terminations, truncations, infos = env.step(acts)
143
144
                  for role in env.agents:
145
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]
                   # duration is the number of milliseconds between frames; this is 40 frames per second
157
                   model_dir = model_path.split(".")[0]
imgs[0].save(model_dir + f"_eval_run_{i}.gif", save_all=True, append_images=imgs[1:],
158
159
                         duration=500, loop=0)
160
161
          env.close()
162
          avg_reward = np.mean(total_rewards)
std_reward = np.std(total_rewards)
163
164
165
          with open(model_path + ".txt", "a") as f:
166
              f.writelines(f"\n\n{env_kwargs}\n")
f.writelines(f"\t Avg reward: {avg_reward}, std: {std_reward}\n")
f.writelines(f"\t Rewards: {total_rewards}\n")
167
168
169
170
          print(f"\t Avg reward: {avg_reward}, std: {std_reward}")
171
172
          return avg_reward
173
174
     if __name__ == "__main__":
175
176
          with open("Experiment/train_agents.yaml", "r") as stream:
177
178
               try:
179
                   experiments = yaml.safe_load(stream)
               except yaml.YAMLError as exc:
180
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

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
   # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
   from __future__ import annotations
16
    import os
17
    import time
    import sys
18
19
   import glob
    from tqdm import tqdm
    import numpy as np
22
    from PIL import Image
23 import yaml
24
25 sys.path.insert(0, "../../")
   from utilities.PZWrapper import PettingZooEnv
    from examples.MAInspection.Environments.MAIcoach import MAI_COACH
  from examples.MAInspection.Environments import env as Multinspect
30 from env import COACH_PettingZoo
    from stable_baselines3 import PPO
   from stable_baselines3.ppo import MlpPolicy
    from stable_baselines3.common.callbacks import EvalCallback
35
    import supersuit as ss
36
37
    import torch
38
    CORES = 6
    torch.set_num_threads(CORES)
39
40
    torch.set_num_interop_threads(CORES)
41
42
    import logging
logging.basicConfig()
43
    logging.getLogger().setLevel(logging.INFO)
44
45
    logger = logging.getLogger(__name__)
46
    pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
47
        name.startswith("pymunk")]
48
    for log_handler in pymunk_loggers:
        log_handler.setLevel(logging.INFO)
49
50
51
    # SB code adapted from https://pettingzoo.farama.org/tutorials/sb3/waterworld/
52
53
    ## We're going to train two policies, one that assumes a densely poisonous env
54
    ## and one that requires sparse exploration.
55
    # %%
56
    def train(
        env_fn,
57
58
        steps: int = 10_{-000},
59
        seed: int | None = 0,
60
        num_vec_envs=1,
61
        num_cpus=1,
62
        learning_rate=1e-3,
63
        batch_size=256,
64
        model_dir=""
65
   ):
66
        os.makedirs(model_dir,exist_ok=True)
67
68
        # Train a single model to play as each agent in a cooperative Parallel environment
69
        env = env_fn()
70
        env.reset(seed=seed)
71
        print(env.possible_agents)
72
        logger.info(f"Starting training on {str(env.metadata['name'])}.")
73
        env = ss.pettingzoo_env_to_vec_env_v1(env)
76
        env = ss.concat_vec_envs_v1(env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
             base_class="stable_baselines3")
```

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

```
156
                  imgs[0].save(model_dir + f"_eval_run_{i}.gif", save_all=True, append_images=imgs[1:],
                       duration=500, loop=0)
157
158
          env.close()
159
160
          avg_reward = np.mean(cum_rewards)
         std_reward = np.std(cum_rewards)
161
162
          with open(model_path + ".txt", "a") as f:
163
              f.writelines(f"\n\n{env_kwargs}\n")
164
             f.writelines(f"\t Avg reward: {avg_reward}, std: {std_reward}\n")
f.writelines(f"\t Rewards: {cum_rewards}\n")
165
166
167
168
         logger.info(f"\t Avg reward: {avg_reward}, std: {std_reward}")
169
         return avg_reward
170
171
     def get_env():
172
173
         return PettingZooEnv(PZGame=Multinspect)
174
     if __name__ == "__main__":
175
          WAYPOINTER_DIR = "waypointer"
176
         WAYPOINTER_PATH = min(glob.iglob('waypointer/*'), key=os.path.getctime)
177
178
          WAYPOINTER_PATH = min(glob.iglob(os.path.join(WAYPOINTER_PATH, '*.zip')), key=os.path.getctime)
179
180
          with open(os.path.join("Experiment", "train_director.yaml"), "r") as stream:
181
182
              try:
183
                  params = yaml.safe_load(stream)
184
              except yaml.YAMLError as exc:
                  print(exc)
185
186
         for agent, param in params["COACH_params"]["Agents"].items():
    param["params"]["policy_path"] = WAYPOINTER_PATH
187
188
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
         model_path = min(glob.iglob(os.path.join('director', '*.zip')), key=os.path.getctime)
205
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:
              with open(os.path.join("Experiment", "dash_app.yaml"), 'w') as g:
214
215
                  for line in f.readlines():
                       line = line.replace("<WAYPOINTER_PATH>", WAYPOINTER_PATH)
216
217
                       line = line.replace("<DIRECTOR_PATH>", model_path)
                       g.write(line)
218
219
220 # %%
```

6 examples/MAInspection/Environments

6.1 examples/MAInspection/Environments/MAIagents.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
2
3
    # Licensed under the Apache License, Version 2.0 (the "License");
      you may not use this file except in compliance with the License.
      You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
   # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   # See the License for the specific language governing permissions and
   # limitations under the License.
14
15
   import numpy as np
16
   import copy
17
    import logging
    from gymnasium.spaces import Box
    logger = logging.getLogger(__name__)
   from numpy.random import PCG64DXSM, Generator
   from agents import *
23
    class WaypointInterface(TrivialInterface):
24
        def __init__(self, role, env):
25
            super().__init__(role, env)
26
            desc = ["rel frame x", "rel frame y", "rel frame z"]
27
28
            policies = ActionBox(
29
30
                {\tt 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
            self.action_dictionary = {"Waypoint": policies}
36
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]
            self.policy = PPO.load(policy_path)
51
52
            self.next_waypoint = None
54
        def __str__(self):
55
            return f"SB_PPOWaypointActor: {self.role}"
56
        def get_action(self, obs, t, mean_mode=False):
58
             bad_command = False
            acting = False
            coa_done = False
60
            action = copy.copy(self.none_action)
63
            if t in self.coa.timeline.keys():
                 ## Start new waypoint
                 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
71
            if self.next_waypoint is not None:
                 i = self.env.possible_agents.index(self.role)
                 pos = self.env.orb[self.role].r.to(self.env._OU_DIS).value
                 pos_c = self.env.orb["chief"].r.to(self.env._OU_DIS).value
75
```

```
76
                   if self.env.OBSERVATION_FRAME == "Hills":
 77
 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
                   agent_obs = np.concatenate([obs[:6*self.env.num_deputies],rel_waypt.reshape(-1)])
if np.linalg.norm(rel_waypt) < self.env._WAYPOINT_ARRIVAL_PROX.value:</pre>
 84
 85
 86
                       acting = False
                       coa_done = True
 87
 88
                       self.next_waypoint = None
 89
                   else:
 90
                       acting = True
 91
                       action, _ = self.policy.predict(agent_obs, deterministic=True)
 92
93
              return action, {
                   "acting": acting,
"coa_done": coa_done,
 94
95
 96
                   "bad_command": bad_command,
              1
97
98
          def reset(self, env):
99
100
              super().reset(env)
101
              self.time_remaining = -1
              self.current_policy = None
102
103
104
              self.interface = SB_PPOWaypointActor.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
     Interfaces = {}
123
     for c in classes:
124
          # BasicActor
          if issubclass(c, TrivialInterface):
125
              Interfaces[c] = []
126
127
128
     Agents = {}
129
     for c in classes:
130
          # BasicActor
          if issubclass(c, BasicActor):
    Interfaces[c.InterfaceType].append(c)
131
132
133
              Agents[c.__name__] = c
```

6.2 examples/MAInspection/Environments/MAIcoach.py

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

```
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)"""
         u, v = np.mgrid[0 : 2 * np.pi : resolution * 2j, 0 : np.pi : resolution * 1j]
 84
         X = radius * np.cos(u) * np.sin(v) + x
Y = radius * np.sin(u) * np.sin(v) + y
 85
 86
         Z = radius * np.cos(v) + z
 87
         return (X, Y, Z)
 88
 89
 90
 91
     def plotly_fig2array(fig):
          # convert Plotly fig to an array
 92
         fig_bytes = fig.to_image(format="png")
 93
 94
         buf = io.BytesIO(fig_bytes)
         img = Image.open(buf)
 95
 96
         return np.asarray(img)
 97
98
99
     class MAI_Trajectory(Trajectory):
100
          def __init__(self, env, initial_return, intial_frame=None):
101
              self.pos = []
102
              self.ori = []
              self.pts = []
103
104
              self.unobserved_points = []
105
              self.cumm_rewards = []
106
107
              super().__init__(env, initial_return, intial_frame=intial_frame)
108
109
         def add(self, env, action, step_return, agent_info, frame=None):
              super().add(env, action, step_return, agent_info, frame=frame)
frame = env.hills_frame(env.orb["chief"])
110
111
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)
              self.ori.append(copy.copy(env.ori) @ frame)
119
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
     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,
              comm_schedule:Timeline=None,
134
135
              seed=6234235,
              TrajectoryClass = MAI_Trajectory,
136
137
138
              super().__init__(env_creator,
139
                  parameter_generator,
140
                  agents=agents,
                  fill random=fill random.
141
142
                  comm_schedule=comm_schedule,
143
                  seed=seed,
                  TrajectoryClass = TrajectoryClass
144
145
146
147
         def plot_trajectory_component(
148
              self,
149
              traj,
150
              coas=None,
151
              env=None,
              plan=None
152
              render_mode="plotly",
153
154
              \verb|simulate_data=False|
155
         ):
156
              if plan:
                  coas = plan.coas
157
158
159
              players = traj.players
```

```
160
              cmap = plt.get_cmap("tab10")
161
              event_cmap = plt.get_cmap("Set1")
              player_to_color = {p: i / 10 for i, p in enumerate(players)}
162
163
              game_len = len(traj)
num_points = env.num_points
164
165
166
              chief_perim = env._CHIEF_PERIMETER
167
              # 3 dim: pos, 4 dim: color
xs = np.zeros([game_len, len(players), 7])
168
169
170
              xs[:,:,:3] = np.array([pos.to(env._OU_DIS).value for pos in traj.pos])
171
172
              # Plot Cone Information
              CONE FREQ = 5
173
              CONE_OPATICY = 0.5
174
175
              n_cones = int(np.ceil(game_len/CONE_FREQ))
176
177
              cones = np.zeros([n_cones*len(players), 10])
178
179
              \mbox{\tt\#} Stack is going to concatenate long the first axis, combine all
180
              # players info into one stack
              cones[:, :3] = np.vstack(xs[::CONE_FREQ,:,:3])
181
              cones[:, 3:6] = np.vstack(np.array(traj.ori)[::CONE_FREQ,:,:])[:,0,:]
182
183
              pcolors = [cmap(player_to_color[p]) for p in players]
              cones[:, 6:] = np.repeat(np.array(pcolors),n_cones, axis=0)
cones[:, -1] = CONE_OPATICY
184
185
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]
                                event_time_left = event.parameters[0]
202
203
                               print(event.label)
204
                                c = event_cmap(event.id)
205
                               print(c)
206
207
                       if event_time_left <= 0:</pre>
208
                           c = cmap(player_to_color[p])
209
210
                       xs[i,p_idx,3:] = c
                       event_time_left -= 1
211
212
213
              ## Get Seen Points
214
              pts = traj.pts[0]
215
              u_set = traj.unobserved_points[0]
216
              unseen = np.zeros(pts.shape[0], dtype=bool)
unseen[list(u_set)] = True
217
218
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
                  {\tt traces.append(plot\_points(unseen, "\#000000"))}
228
              if seen.shape[0] > 0:
229
                  traces.append(plot_points(seen, "#FFFFFF"))
230
231
232
              # # Draw Chief Sphere
233
              (x_pns_surface, y_pns_surface, z_pns_suraface) = 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_suraface,
239
                       opacity=0.5,
240
                       showscale=False,
```

```
241
                )
242
243
244
            # View Cones:
245
            traces.append(
246
                go.Cone(
247
                    x=cones[:, 0],
                    y=cones[:, 1],
248
249
                    z=cones[:, 2],
250
                    u=cones[:, 3],
                    v=cones[:, 4],
251
252
                    w=cones[:, 5],
253
                    opacity=0.1,
254
                    showscale=False.
255
                    showlegend=False,
                    sizemode="absolute",
256
257
                    sizeref=10.
                    anchor="tip",
258
259
                )
            )
260
261
            fig = go.Figure(data=traces)
262
            fig.update_layout(
263
                scene_aspectmode="cube",
264
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":
                return plotly_fig2array(fig)
289
290
291
292
    293
    ## Example Usage
294
    295
    if __name__ == "__main__":
296
297
        from examples.MAInspection.env import MultInspect
298
299
        ## Environemntal Parameters
        env_params = {"_OBS_REWARD":.01, "num_deputies": 2}
300
301
302
        # Actor Parameters
303
        COACH_params = {
             "Agents": {
304
                "player_0": {
305
                     "class_name":"SB_PPOWaypointActor",
306
                    "params": {"policy_path":
307
                         "/root/coach/examples/MAInspection/waypointer/MAInspect_20240205-234515.zip"}
                         If the agent has setup parameters
308
                    },
                "player_1": {
309
                    "class_name": "SB_PPOWaypointActor",
310
                    "params": {"policy_path":
311
                         "/root/coach/examples/MAInspection/waypointer/MAInspect_20240205-234515.zip"}
                         If the agent has setup parameters
312
                    },
313
            }
        }
314
315
316
         # Create actors from parameters
317
         agent_dict = dict()
```

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

6.3 examples/MAInspection/Environments/env.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
2
3
   # Licensed under the Apache License, Version 2.0 (the "License");
   # you may not use this file except in compliance with the License.
   # You may obtain a copy of the License at
        http://www.apache.org/licenses/LICENSE-2.0
   \mbox{\tt\#} Unless required by applicable law or agreed to in writing, software \mbox{\tt\#} distributed under the License is distributed on an "AS IS" BASIS,
10
   # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   # See the License for the specific language governing permissions and
13
   # limitations under the License.
14
   # %%
15
   """Multi-Inspection Environment"""
16
17
18
   import argparse # for type hinting
19
   import copy
20
   import logging
   from collections.abc import Callable # for type hinting
22
   from typing import Any, Optional, TypeVar, Union, cast, NewType
   import gymnasium as gym
   import imageio
   import matplotlib # type: ignore[import]
   import matplotlib.lines as mlines # type: ignore[import]
   import matplotlib.pyplot as plt # type: ignore[import]
   import numpy as np
    # from pettingzoo.utils.env import ParallelEnv
   from matplotlib.backends.backend_agg import (
33
       FigureCanvasAgg as FigureCanvas, # type: ignore[import]
34
   from matplotlib.collections import LineCollection
35
   from mpl_toolkits.mplot3d.art3d import Line3DCollection # type: ignore[import]
   from numpy.random import PCG64DXSM, Generator
   from scipy.spatial.transform import Rotation # type: ignore[import]
38
   from scipy.stats import multivariate_normal, ortho_group # type: ignore[import]
40
   import sys
41
42
43 \quad \hbox{from numpy.typing import NDArray}
   FloatArray = NDArray[np.float_]
44
   RenderFrame = NDArray[np.uint8]
45
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
    def get_env_class(args: argparse.Namespace) -> Callable[..., Any]:
61
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
        the class to use in creating env objects """
73
        return MultInspect
   def unit(v):
        if len(v.shape) == 1:
           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
     def frame(r, v):
 84
 85
        r = unit(r)
         v = unit(v)
 86
 87
        n = np.cross(r, v, axis=1)
 88
 89
         return np.stack([r,v,n],axis=2)
 90
     def parallel_env(**kwargs):
 91
92
         return MultInspect(**kwargs)
 93
     class MultInspect(ParallelEnv):
 94
 95
96
         We assuime 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 vcetor 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 \mbox{->} v_x,
110
         M_2^T: v_x \rightarrow v_f^2, so M_2^TM_1 * v_f^1 = v_f^2 is a appropreate 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
118
         Attributes
119
120
         agents: list[AgentID]
121
122
            A list of the names of all current agents, typically integers. May changed as environment
                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
                 in the observation and action spaces. This cannot be changed through play or resetting.
127
         max_num_agents: int
128
            The length of the possible_agents list.
129
         observation_spaces: dict[AgentID, gym.spaces.Space]
130
            A dict of the observation spaces of every agent, keyed by name. This cannot be changed through
                play or resetting.
131
         action spaces: dict[AgentID, gvm.spaces.Space]
             A dict of the action spaces of every agent, keyed by name. This cannot be changed through play
132
                 or resetting.
133
134
135
         Methods
136
137
                                                tuple[dict[str, ObsType], dict[str, float], dict[str,
138
         step(actions: dict[str, ActionType])
             bool], dict[str, bool], dict[str, dict]]
139
             Receives a dictionary of actions keyed by the agent name.
             140
                 dictionary and info dictionary, where each dictionary is keyed by the agent.
141
         reset(seed: int | None = None, options: dict | None = None)
                                                                        dict[str, ObsType]
142
             Resets the environment.
         seed(seed=None)
143
144
            Reseeds the environment (making it deterministic).
145
         render() None | np.ndarray | str | list
146
            Displays a rendered frame from the environment, if supported.
147
         close()
148
            Closes the rendering window.
149
         state()
                    ndarray
150
            Returns the state.
151
         observation_space(agent: str)
                                         Space
152
            Returns the observation space for given agent.
153
         action_space(agent: str)
                                     Space
```

```
154
155
156
         metadata = {"render_modes": ["human", "rgb_array"], "render_fps": 50, "name": "MAInspect"}
157
158
         def __init__(self, render_mode="rgb_array", **kwargs) -> None:
159
160
              self.augment(params = kwargs)
161
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 appropreate key exists.
170
             ## The list below provides the default parameters
171
172
              self.verbose: bool = False
              self.six_axis: bool = True
173
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
             # Compatability
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.ARGLAT = 0 << u.deg ## Latitude updown
195
              self.INC = 0 << u.deg ## Direction of orbit
196
              self._STARTING_DISTANCE_FROM_CHIEF = 150 << u.m
197
              self.offset_angle = (1/10)*(self._STARTING_DISTANCE_FROM_CHIEF /
                  self.INIT_ALT).decompose().value << u.rad</pre>
199
200
              # Action Frame
              self.ACTION_FRAME = "Hills"
201
                                                    # Options should be Hills and Orientation
202
              self.OBSERVATION_FRAME = "Hills"
                                                    # Options should be Hills and Orientation
              self._OU_DIS = u.m
203
204
              self._OU_TIM = u.s
205
              self._OU_VEL = self._OU_DIS/self._OU_TIM
206
207
              self._CHIEF_PERIMETER: float = 50 << u.m
208
              self._DEPUTY_PERIMETER: float = 150 << u.m</pre>
209
              self._DEPUTY_MASS: float = 1 << u.kg
210
              self._DEPUTY_RADIUS: float = 1 << u.m
211
              self._DEPUTY_THRUST_COEEF: float = 0.01
212
213
              self._SIM_OUTER_PARAMETER: float = 200 << u.m #
              self._MAX_OUTER_PERIMETER: float = 300 << u.m # If you leave here, you truncate</pre>
214
              self._STARTING_VEL_NORM: float = 0 << u.m/u.s # 0.001
215
216
              self._TIME_PER_STEP: float = 90 << u.s
217
             self._TAU_per_THRUST: float = 90 << u.s
self._DELTA_T: float = 0.1 << u.m/u.s
self._DELTA_T: float = 90 << u.s
self._TAU_per_THRUST: float = 0.1
218
219
220
              self._USE_ANGULAR_MOMENTUM = False
221
222
              self._OBS_REWARD: float = 0.02 # 20
223
              self._REWARD_FOR_SEEING_ALL_POINTS: float = 1 # 200
224
              self._CRASH_REWARD: float = 0
self._REWARD_PER_STEP: float = 0.0 # -.1 # Reward per tick
225
226
              self._REWARD_FOR_LEAVING_PARAMETER: float = 0
227
228
              self._REWARD_FOR_LEAVING_OUTER_PERIMETER: float = 0
229
230
              self._SOLID_CHIEF: bool = True
231
              self. MAX BURN: float = 10
232
233
              self._REW_COV_MTRX = 300**2
```

```
234
             self._PROX_RWD_SCALE = 100000
235
236
             # Visualization
             self._VIS_NUM_CONE_LINES = 8
237
238
             self._VIS_CONE_LEN = 40 << u.m
239
240
             # Vision Cone:
             self._MIN_VISION: float = 0 << u.m
241
             self._MAX_VISION: float = np.inf << u.m
self._VISION_ARC: float = np.pi / 8 << u.rad
242
243
244
245
             # Apply parameters from input
246
             \verb|self._apply_parameters(params=env_params)|\\
247
             if hasattr(env_params, "args"):
                 self.master_seed: int = env_params.args.master_seed
248
249
             else:
250
                 self.master_seed: int = 487924
251
             self.seed(seed=self.master_seed)
252
253
             if self._TRAIN_WAYPOINTER:
254
255
                 self.num_points = self.num_deputies
256
257
             # Check parameters from input
             assert self._MIN_VISION >= 0, "_MIN_VISION must be >= 0"
258
             assert self._MAX_VISION > self._MIN_VISION, "_MAX_VISION must be > _MIN_VISION"
259
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
                      -self._MAX_OUTER_PERIMETER.value
280
281
                      * np.ones(3 * self.num_deputies), # Deputy velocity
282
283
                      * self._MAX_OUTER_PERIMETER.value
                      * np.ones(3 * (self.num_deputies - 1)), # Dep. Positions, except me
284
285
286
                      * self._MAX_OUTER_PERIMETER.value
287
                      * np.ones(3), # Single Waypoint,
288
                 1
289
290
                 raw_high_values: list[float] = [
                      self._MAX_OUTER_PERIMETER.value * np.ones(3), # Chief Position
291
292
                      self._MAX_OUTER_PERIMETER.value
293
                      * np.ones(3 * self.num_deputies), # Deputy velocity
294
295
                      * self._MAX_OUTER_PERIMETER.value
                      * np.ones(3 * (self.num_deputies - 1)), # Dep. Positions, except me
296
297
                      * self._MAX_OUTER_PERIMETER.value
298
299
                      300
                 1
301
             else:
                 raw_low_values: list[float] = [
302
                      -self._MAX_OUTER_PERIMETER.value * np.ones(3), # Chief Position
303
304
                      -self._MAX_OUTER_PERIMETER.value
305
                      * np.ones(3 * self.num_deputies), # Deputy velocity
306
                      * self._MAX_OUTER_PERIMETER.value
* np.ones(3 * (self.num_deputies - 1)),  # Dep. Positions, except me
307
308
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] = [
                       self._MAX_OUTER_PERIMETER.value * np.ones(3), # Chief Position
316
317
                       self._MAX_OUTER_PERIMETER.value
318
                       * np.ones(3 * self.num_deputies), # Deputy velocity
319
320
                       * self._MAX_OUTER_PERIMETER.value
321
                       * np.ones(3 * (self.num_deputies - 1)), # Dep. Positions, except me
322
323
                       * self._MAX_OUTER_PERIMETER.value
                       * np.ones(3 * (self.num_points)), # Point Pos.
324
                       np.ones(self.num_points),
325
326
327
              low = np.concatenate(raw_low_values, dtype=np.float32)
328
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:
                  low = np.array([-10, -10, -10, -np.pi, -np.pi, -np.pi], dtype=np.float32)
334
335
                  act_space = gym.spaces.Box(low=low, high=-low, dtype=np.float32)
336
                  # Thrust, rotate Left/Right, rotate UP/Down
act_space = gym.spaces.Box(
337
338
                       low=-np.ones(3, dtype=np.float32),
high=np.ones(3, dtype=np.float32),
339
340
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
                   earth frame
350
              self.rot = np.zeros([self.num_deputies, 3])
                                                                         # Depute Angular Momentum
351
352
              self.pos = np.zeros([self.num_deputies, 3])
                                                                         # Depute Position Holder
353
              self.vel = np.zeros([self.num_deputies, 3])
                                                                         # Depute Velocity Holder
354
                                                                        # Current frame for the cheif relative to
355
              self.chief_frame = np.zeros([3, 3])
                  absolute earth frame
356
              self.frames = np.zeros([self.num_deputies, 3, 3])
                                                                         # Current frame for the deputies relative
                  to absolute earth frame
357
              self.pts = np.zeros([self.num_deputies, 3])
                                                                         # Chief Inspection Points
              self.nor = np.zeros([self.num_deputies, 3])
                                                                        # Chief Inspection Normals
358
359
360
              self.sim_steps = 0
361
362
              self.unobserved_points = set(range(self.num_points))
363
364
              self.render_path: Optional[str] = None
              self.frame_store: list[RenderFrame] = []
self.cum_rewards = np.zeros(self.num_deputies)
365
366
367
368
          def _apply_parameters(self, params: dict) -> None:
              for k, v in params.items():
    self.__dict__[k] = copy.copy(v)
369
370
371
          def augment(self, params: dict) -> None:
372
373
              self._setup_gym_env(env_params=params)
374
          def seed(self, seed: int) -> None:
    self.np_random = Generator(PCG64DXSM(seed=seed))
375
376
377
          def observation_space(self, agent: Role) -> gym.spaces.Box:
378
379
              return self.observation_spaces[agent]
380
381
          def action_space(self, agent: Role) -> gym.spaces.Box:
382
              return self.action_spaces[agent]
383
384
          ## Reset
385
          def reset (
386
              self.
387
              *,
388
              seed: Optional[int] = None,
              options: Optional[dict[Any, Any]] = None,
render_path: Optional[str] = None,
389
390
         ) -> tuple[dict[Role, Any], dict[Role, dict[str, Any]]]:
391
392
```

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

```
self.pos = np.stack([self.orb[role].r-self.orb["chief"].r for role in self.possible_agents])
self.vel = np.stack([self.orb[role].v-self.orb["chief"].v for role in self.possible_agents])
473
474
475
476
              # Compute Which Points Seen
477
             just_seen = self._detect_points()
478
479
              # Compute rewards
             if self._TRAIN_WAYPOINTER:
480
                  self._compute_waypoint_reward(just_seen)
481
482
                  obs = self._make_waypoint_observations()
483
              else:
484
                  self._compute_reward(just_seen)
485
                  obs = self._make_observations()
486
              # Check if all of the deputes have truncated.
487
488
             if all(self.truncated.values()):
489
                  terminated = True
490
             if self.render_path:
491
492
                  frame = self.render()
493
                  assert frame is not None
494
                  self.frame_store.append(frame)
495
496
                  if terminated:
                      imageio.mimwrite(
497
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
              self.obs = obs
506
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
             self.terminated = {agt: self.terminated_this_step for agt in self.possible_agents}
513
514
515
              # Remove Terminated and Truncated agents from avaiable 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
525
              # Set the internal active players array
526
              self._active = np.array([role in self.agents for role in self.possible_agents])
527
528
             return (self.obs, self.reward, self.terminated, self.truncated, self.info)
529
530
         def _compute_reward(self, just_seen: list[list[int]]) -> float:
531
532
              self.sra = np.zeros(self.num_deputies) # Step Reward Array
              self.step_rewards = {role: self.sra[i:i+1] for i, role in enumerate(self.agents)} # Slices are
533
                  pointers to the reward array
534
535
              self._insepct_reward(just_seen)
536
              self._chief_prox_reward()
537
              self._game_length_exceed()
538
              self._seeing_all_points()
539
              self._leaving_outer_perimeter()
540
541
         def _compute_waypoint_reward(self, just_seen: list[list[int]]) -> float:
542
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
                  pointers to the reward array
545
546
              self._go_towards_waypoint()
547
              self._game_length_exceed()
548
              self._leaving_outer_perimeter()
549
              self._chief_prox_reward()
550
```

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
                  if np.linalg.norm(self.pos[i] - self.pts[i]) < self._WAYPOINT_ARRIVAL_PROX: self.step_rewards[role] += self._WAYPOINT_ARRIVE_REWARD
556
557
                      self.terminated_this_step = True
558
559
560
561
562
         def _seeing_all_points(self):
563
              if len(self.unobserved_points) == 0:
                  R = (self.max_episode_steps - self.sim_steps) / self.max_episode_steps
564
                  self.sra += R * self._REWARD_FOR_SEEING_ALL_POINTS
565
                  self.terminated_this_step = True
566
567
568
569
570
         def _leaving_outer_perimeter(self):
              I = np.where(np.linalg.norm(self.pos, axis=1) > self._MAX_OUTER_PERIMETER)[0]
571
572
              # print(np.linalg.norm(self.pos, axis=1) > self._MAX_OUTER_PERIMETER)
573
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]
                  # Initialize reward
598
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:
                      # Reward for seeing points for the first time
reward += len(new_pts) * self._OBS_REWARD
607
608
                      # Update unseen points
609
                      self.unobserved_points = self.unobserved_points.difference(new_pts)
610
611
                  self.step_rewards[role] += reward
612
613
614
615
616
         {\tt def \_chief\_prox\_reward(self):}
              # Are we active and outside the safe zone?
617
              outside_safe_zone = self._active & (np.linalg.norm(self.pos, axis=1) > self._SIM_OUTER_PARAMETER)
618
619
              # Are we active and inside the chief?
              inside_chief = self._active & (np.linalg.norm(self.pos, axis=1) < self._CHIEF_PERIMETER)</pre>
620
621
              # Adjust reward
622
              self.sra[outside_safe_zone] += self._REWARD_FOR_LEAVING_PARAMETER
623
624
              self.sra[inside_chief] += self._CRASH_REWARD
625
             # Kill agents inside chief
626
627
              self._active[inside_chief] = False
628
629
              for i in np.where(inside_chief)[0]:
630
                  self.terminated_this_step = True
631
```

```
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
         # Physics goes here
642
         def _propagate_objects(self, actions):
643
               Poliastro assumes earth centered coordinates. It fixes an x,y, and z. So
644
645
              # vectors in poliastro objects are in terms of of an absolute coordinate system.
646
             # Translate actions into action frame and apply impulses
647
648
649
             for player, (role, act) in enumerate(actions.items()):
650
                  # Compute Velocity Change
                 dv = act[:3].reshape(-1,1) * self._DELTA_V_PER_THRUST
if self.ACTION_FRAME == "Hills":
651
652
                      frame = self.hills_frame(self.orb[role])
653
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
                  \mbox{\tt\#} 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
669
                  if self._USE_ANGULAR_MOMENTUM:
670
                      self.rot[player] += tau
                      tau = self.rot
671
672
673
                  R = Rotation.from_euler("xyz", tau)
                  T = frame @ R.as_matrix() @ frame.T # x columns to frame, rotate, then back to x
674
675
676
                  self.ori[player, :, :] = T @ self.ori[player, :, :]
677
678
              # Rotate Chief Points:
679
             if self.chief_angular_momentum is not None:
                 R = self.chief_angular_momentum
680
681
                  self.pts = R @ self.pts
682
683
              # Propagate Orbits
684
              for role, orb in self.orb.items():
685
                  self.orb[role] = orb.propagate(self._TIME_PER_STEP)
686
687
              # for role in self.possible_agents:
688
                   print(role, self.orb[role].r, np.linalg.norm(self.orb[role].r - self.orb["chief"].r))
689
690
         def _detect_points(self):
691
              just_seen = []
692
              for player, role in enumerate(self.possible_agents):
693
                  # Position relvative to chief
694
                  pos = self.orb[role].r - self.orb["chief"].r
695
696
                  # Detect Points:
697
                  # This handles occlusion by the chief
                 \mbox{\tt\#} Basically, what points are on the same side of the chief as you see
able_points = np.where(
698
699
                      (self.pts * (pos - self.pts)).sum(axis=1) > 0
700
                 101 (
701
702
                 #########
703
704
                  # Spherical Vision
705
                  ########
                  # point holder
706
707
                  seen_points: list[int] = []
708
709
                  # loop over possible points
710
                  for point in seeable_points:
711
                      # Cone References
712
                      # cone:
                           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
# The initial distance calc though is fine, and easy
714
715
716
                                          # Shell References
                                               This is a pain in the a$$ - I hate math vs physics
717
718
                                               Used all of these references to generate a consensus algorithm
                                               https://en.wikipedia.org/wiki/Spherical_coordinate_system
719
                                                 This one doesn't fully work
720
721
                                              https://en.wikipedia.org/wiki/Atan2
                                                 This explains why atan2 vs atan
722
723
                                          # https://mathworld.wolfram.com/SphericalCoordinates.html
724
                                                 https://stackoverflow.com/questions/4116658/faster-numpy-cartesian-to-spherical-coordinate-conversion\\ THIS IS MY FAVORITE ONE - uses physics notation
725
726
                                                  https://math.libretexts.org/Bookshelves/Calculus/Calculus_(OpenStax)/12\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12.07\%3A\_Vectors\_in\_Space/12
727
                                                 This supports the stackoverflow, just in math notation
728
729
                                          # Position relative to agent
730
                                          v = self.pts[point, :] - pos
731
732
                                          # Distance relative to agent
                                          \mbox{\tt\#} this is the same as the distance in the deputy frame, but
733
734
                                          # doesn't require the transformation
735
                                          p_dist = np.linalg.norm(v)
736
737
                                          # Check if distance is within shell
738
                                          # Ignoring angles at the moment
739
                                          # Less than max, more than min
740
                                                 I assume we're usually too far away
741
                                          if (p_dist < self._VISION[role][2]) and (</pre>
742
                                                 p_dist >= self._VISION[role][1]
743
744
                                                  # Orient point in deputy frame
745
                                                 pdf = self.ori[player].T @ v
746
747
                                                  # theta and phi
748
                                                  # This is physics notation
749
                                                  # theta = polar/zenith angle, [0, pi], z-axis is 0
750
                                                       phi = azimuth angle, (-pi, pi], x-axis is 0
751
                                                 theta = np.arccos(pdf[2] / p_dist)
752
                                                 phi = np.arctan2(pdf[1], pdf[0])
753
754
                                                  \# theta needs to measure from x-axis, not z-axis
755
                                                 # calculate the complementary angle
                                                 # theta = [pi/2, -pi/2], x-axis is 0 theta = (np.pi / 2 << u.rad) - theta
756
757
758
759
                                                  # Check if point is within cone
                                                  # Assume circular directions, so abs() it
760
                                                 # NOTE: We could implement different ranges for theta and phi
761
762
                                                                 This would give a non-circular viewing angle
763
                                                 if (np.abs(phi) < self._VISION[role][0]) and (
764
                                                         np.abs(theta) < self._VISION[role][0]
765
766
                                                         # vou can see it!
767
                                                          seen_points.append(point)
768
769
                                  just_seen.append(seen_points)
770
                                  if self.verbose:
                                          print(f"{player} seeable: {seeable_points}, seen: {seen_points}")
771
772
                         return just_seen
773
774
                  def accel(self, t0, state, k, rate=1e-5):
775
                            ""Constant acceleration aligned with the velocity. """
776
777
                          v vec = state[3:]
778
                          #v_vec = state[:3]
                         norm_v = (v_vec * v_vec).sum() ** 0.5
return -rate * v_vec / norm_v
779
780
781
782
                  def f(self, t0, u_-, k):
783
                         \# t_0: time to evaluate at
                         # u_- = [x,y,z,vx,vy,vz] in earth coords, rather annoyingly unitless. # Assumed units (I've tested this) are km and km/s
784
785
786
787
                         # U.append(u_)
788
                          du_kep = func_twobody(t0, u_, k)
789
                         \#ax, ay, az = self.accel(t0, u_, k, rate=1e-5)
                         ax, ay, az = self.acc
du_ad = np.array([0, 0, 0, ax, ay, az])
790
791
```

```
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)
              \tt return frame(r,v_p)[0].decompose().value
798
799
          def _make_observations(self, obs_frame=None) -> dict[Role, FloatArray]:
800
801
              # Return dictionary
802
803
              # Play role will be keys, observations will be values
804
              obs = dict()
805
806
              if obs frame is None:
                  obs_frame = self.OBSERVATION_FRAME
807
808
809
              # Setup point mask
              # hide points that have been observed
810
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
              pos_c = self.orb["chief"].r.reshape(-1,1).to(self._OU_DIS).value
vel_c = self.orb["chief"].v.reshape(-1,1).to(self._OU_VEL).value
817
818
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 multiplacation
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
831
                       frame = self.ori[i]
832
833
                  vel_in_frame = frame.T @ (vels - vels[:,[i]])
                  chief_vel_in_frame = frame.T @ (vel_c - vels[:,[i]])
834
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
                  rel_pos = (
840
841
                       frame.T @ np.delete(arr=poss - poss[:,[i]], obj=i, axis=1)
842
843
844
                  dchief = pos_c - poss[:,[i]]
845
846
                  # Relative positions of all points in my frame
847
                  rel_pts = frame.T @ (pos_p + dchief)
848
                  # Relative position of chief in my
rel_chief = frame.T @ dchief
849
850
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
                  # Store observation space under appropriate role
obs[Role(f"player_{i}")] = np.concatenate([tmp.reshape(-1), pt_mask])
863
864
865
866
              return obs
867
868
          def _make_waypoint_observations(self, obs_frame=None) -> dict[Role, FloatArray]:
869
              # Return dictionary
870
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
              observedPoints = list(set(range(self.num_points)) - self.unobserved_points)
879
              pt_mask = np.ones(self.num_points)
880
881
              pt_mask[observedPoints] = 0
882
              poss = np.stack([self.orb[role].r for role in self.possible_agents]).T.to(self._OU_DIS).value
vels = np.stack([self.orb[role].v for role in self.possible_agents]).T.to(self._OU_VEL).value
883
884
885
              pos_c = self.orb["chief"].r.reshape(-1,1).to(self._OU_DIS).value
              pos_c = self.orb["chief"].T.Issnape(-1,1).to(self._OU_VEL).value
pos_p = self.pts.T.to(self._OU_DIS).value
886
887
888
889
              # loop over all deputies
890
              for i, role in enumerate(self.possible_agents):
891
892
                   # Recall: Moving from absolute coords into a frame is left multiplacation
893
                   # by the transpose
894
895
                   # Velocities
                   # Velocities of all deputies in my frame
896
897
                   if obs_frame == "Hills":
898
                       frame = self.hills_frame(self.orb[role])
899
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=poss - poss[:,[i]], obj=i, axis=1)
911
912
913
                   dchief = pos_c - poss[:,[i]]
914
915
                   # print("dchief", dchief)
916
                   # print(self.pts.T.to(self._OU_DIS).value + dchief)
                   # Relative positions of all points in my frame
917
918
                   rel_pts = frame.T @ (pos_p[:,[i]] + dchief)
919
920
                   # Relative position of chief in my
                   rel_chief = frame.T @ dchief
921
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
obs[Role(f"player_{i}")] = tmp.reshape(-1)
935
936
937
              return obs
938
939
940
          ## Visualizations
941
          def render (
942
943
              self.
              render_mode: str = "rgb_array",
944
              vision: Optional[int] = 0, # player number, but not their role, just the number
945
              rotate: bool = False,
elev: float = 45, # I think it's degrees?
946
947
              close: bool = False,
948
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
               poss = np.stack([self.orb[role].r-self.orb["chief"].r for role in
961
               self.possible_agents]).T.to(self._OU_DIS).value
vels = np.stack([self.orb[role].v-self.orb["chief"].v for role in
962
                    {\tt self.possible\_agents]).T.to(self.\_OU\_VEL).value
963
964
               #########
965
               # Title
               #########
966
               # plot scores as title
score_title = " ".je
967
968
                                    ".join(
                    [f"Player {i}: {self.cum_rewards[i]:.3f}" for i in range(self.num_deputies)]
969
970
971
               fig.suptitle(
972
                   t=score_title, horizontalalignment="center", verticalalignment="center"
973
974
975
               #########
976
               # Setup Plot Area
977
               #########
978
               # Plot just the chief, or the chief and one agent-centric view?
979
               if vision is not None:
980
                    ax = fig.add_subplot(141, projection="3d")
981
                else:
982
                   ax = plt.axes(projection="3d")
983
984
               #########
985
               # First Sub Area
986
               #########
987
               # Orientation
988
               deg = 45
989
               if (vision is not None) and rotate:
990
                    x, y = poss[vision, 0:2] / np.linalg.norm(poss[vision, 0:2])
991
                    deg = np.degrees(np.arctan2(y, x))
992
993
               ax.view_init(elev=elev, azim=deg)
994
995
996
               # ax.set_xlim([-300, 300])
 997
               # ax.set_ylim([-300, 300])
               # ax.set_zlim([-300, 300])
 998
999
1000
1001
               # Deputies
1002
               #########
1003
               # I think all of this just plots deputies - JB
1004
1005
               # Deputy body
1006
               for i in range(self.num_deputies):
1007
                    x, y, z = poss[:,[i]]
                    ax.plot3D(x, y, z, marker="^", linewidth=0, label=f"player_{i}")
1008
1009
1010
               # Velocity
               # Plot line collection: LC = np.array([N,S,D]), num lines, num of points
# per line, dim of space, so LC[2,0,:] is point 0 on line 2
1011
1012
1013
1014
1015
               TPS = (self.TIME_PER_STEP * self._OU_VEL/self._OU_DIS).decompose().value
1016
1017
               segs = np.stack([poss.T, poss.T + TPS * vels.T], axis=1)
line_segments = Line3DCollection(
1018
1019
                    segs, linestyle="solid", label="Velocity", color="k"
1020
               ax.add_collection(line_segments)
1021
1022
1023
               # Orientation
               colors = ["r", "g", "b"]
labels = ["Roll/Heading", "Yaw", "Pitch"]
1024
1025
1026
               for axis in range(len(labels)):
1027
                    segs = np.stack([poss.T, poss.T + 30 * self.ori[:, axis, :]], axis=1)
1028
                    line_segments = Line3DCollection(
1029
                        segs, linestyle="solid", color=colors[axis], label=labels[axis]
1030
1031
                    ax.add_collection(line_segments)
1032
```

```
1033
                #########
1034
                # Points on Chief
1035
                #########
1036
                # Shift for plotting
1037
                view_vec = 100 * np.ones(3) / np.sqrt(3)
1038
1039
                # Unobserved points
                if len(self.unobserved_points) > 0:
1040
1041
                    # get positions of points
unobserved_pos = self.pts[list(self.unobserved_points), :]
1042
1043
                    # calc vector for point size
                    s = np.linalg.norm(unobserved_pos.to(self._OU_DIS).value + view_vec, axis=1) - 40
1044
1045
                    # get in euclidean space
1046
                    x, y, z = unobserved_pos.T
1047
                    # plot
                    ax.scatter(x, y, z, label="Unseen", s=s, color="m")
1048
1049
                else:
1050
                    # plot for legend
1051
                    # 60: comes from s, when unobserved_points is the empty set
ax.scatter([], [], [], label="Unseen", s=60, color="m")
1052
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
                    x, y, z = observed_pos.T
1062
1063
                    # plot
1064
                    ax.scatter(x, y, z, color="c", s=s, label="Seen")
1065
1066
                    # plot for legend
                    # 60: comes from s, when observed_points is the empty set
ax.scatter([], [], [], color="c", s=60, label="Seen")
1067
1068
1069
1070
                #########
                # Legend
1071
1072
                #########
1073
                # get legend handles
1074
                # handles, _ = ax.get_legend_handles_labels()
1075
1076
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]
               x = self._CHIEF_PERIMETER * np.cos(u) * np.sin(v)
1083
               y = self._CHIEF_PERIMETER * np.sin(u) * np.sin(v)
1084
                z = self._CHIEF_PERIMETER * np.cos(v)
1085
1086
                ax.plot_wireframe(x, y, z, color="k", alpha=0.2)
1087
               mins = np.min(poss.T, axis=0) - 10
maxes = np.max(poss.T, axis=0) + 10
1088
1089
1090
1091
                widths = []
1092
                centers = []
                for i, s in enumerate(['x', 'y', 'z']):
1093
                    lim = getattr(ax, f"get_{s}lim")()
lim = [
1094
1095
                         min(max(mins[i], -300), -self._CHIEF_PERIMETER.to(self._OU_DIS).value+10),
max(min(maxes[i], 300), self._CHIEF_PERIMETER.to(self._OU_DIS).value+10)
1096
1097
1098
1099
                    \verb|widths.append(lim[1]-lim[0])|\\
1100
                    centers.append((\lim [1]+\lim [0])/2)
1101
1102
                w = max(widths)/2
                for i, s in enumerate(['x', 'y', 'z']):
1103
                    getattr(ax, f"set_{s}lim")([centers[i] - w, centers[i] + w])
1104
1105
1106
               #########
1107
1108
                # Render Agent View
1109
                #########
1110
                \hbox{if vision is not None:}\\
                    ax2 = fig.add_subplot(142, projection="3d")
1111
1112
                    all_obs = self._make_observations(obs_frame="Orientation")
1113
                    role = cast(Role, f"player_{vision}")
```

```
1114
                   obs = all_obs[role]
1115
1116
                   mask = obs[-self.num_points :]
                   obs = obs[:-self.num_points].reshape(3,-1)
1117
1118
                   x = obs[:,-self.num_points:]
1119
1120
                   # Orientation
                   z = np.zeros(2)
1121
1122
                   o = 30 * np.array([0, 1])
                   ax2.plot(o, z, z, color="r")
1123
                   ax2.plot(z, o, z, color="g")
1124
                   ax2.plot(z, z, o, color="b")
1125
1126
1127
1128
1129
                   seen = x[:, mask == 0]
                   unseen = x[:, mask == 1]
1130
                   ax2.scatter(seen[0], seen[1], seen[2], color="c")
1131
                   {\tt ax2.scatter(unseen[0], unseen[1], unseen[2], color="m")}
1132
1133
                   ax2.set_xlim([-100, 200])
                   ax2.set_ylim([-100, 200])
1134
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
              if "frames" not in self.render_data.keys():
1163
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")
              frame = self.hills_frame(self.orb["chief"])
1170
1171
               chief_rel_poss = (frame.T @ poss.T).T
1172
1173
               chief_rel_poss = chief_rel_poss - chief_rel_poss[c_idx,:]
1174
              self.render_data["chief_rel_poss"].append(chief_rel_poss)
self.render_data["frames"].append(frame)
1175
1176
1177
              hills = np.array(self.render_data["chief_rel_poss"])
1178
1179
1180
              for i in range(N):
1181
                   ax4.plot(hills[:, i, 1], hills[:, i, 0])
1182
               \texttt{chief = plt.Circle((0, 0), self.\_CHIEF\_PERIMETER.to(self.\_OU\_DIS).value, color='k', fill=False)}
1183
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
                               def state(self) -> None:
1204
1205
                                           raise NotImplementedError("State Not Implemented.")
1206
1207
1208
                  if __name__ == "__main__":
                               from tqdm import tqdm from PIL import Image
1209
1210
1211
                               env = MultInspect(
1212
                                            num_deputies=4,
_SIM_OUTER_PARAMETER=100<<u.m,</pre>
1213
1214
                                            _CHIEF_PERIMETER = 100 << u.m
1215
1216
                               env.seed(0)
1217
1218
                               env.reset()
1219
                               frames = []
1220
                               # for i in tqdm(range(env.max_episode_steps)):
1221
                               for i in tqdm(range(10)):
1222
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, append\_images=1000, append[1:], duration=500, append[1:], duration=500, append[1:], duration=500, append[1:], duration=500, append[1:], duration=500, append[1:], duration=500, append[1:
                                             loop=0)
1233 # %%
```

7 examples/MAInspection/Experiment

7.1 examples/MAInspection/Experiment/dash_app.yaml

```
2
     Default: # Name of Interface
3
        interface_class: DefaultInterface
       models:
            class_name: DefaultActor
     Waypointer: # Name of Interface
       interface_class: WaypointInterface
11
       roles: [player_0] # If this is not supplied, assumed all roles
       models:
12
13
         PPOWaypointer:
           class_name: SB_PPOWaypointActor
14
15
             policy_path: waypointer/MAInspect_20240213-193621/model.zip
16
18 directors:
19
     PPO_Director:
       class_name: SB3_PP0_Director
20
       path: director/MAInspect_20240206-211327.zip
       roles:
23
        player_0:
           classes: [PPOWaypointer]
```

7.2 examples/MAInspection/Experiment/dash_app_template.yaml

```
interfaces:
   Default: # Name of Interface
2
3
        \verb|interface_class: DefaultInterface|\\
4
        models:
          Default:
5
6
            class_name: DefaultActor
      Waypointer: # Name of Interface
9
        interface_class: WaypointInterface
10
        roles: [player_0] # If this is not supplied, assumed all roles
11
12
        models:
          PPOWaypointer:
13
14
            class_name: SB_PPOWaypointActor
            params:
              policy_path: <WAYPOINTER_PATH>
16
18
   directors:
19
     PPO_Director:
       class_name: SB3_PP0_Director
20
        path: <DIRECTOR_PATH>
^{22}
        player_0:
            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
_PROX_RWD_SCALE: 10000
```

7.4 examples/MAInspection/Experiment/train_director.yaml

```
env_params:
  num_deputies: 1
 1 2
    max_episode_steps: 150
_TRAIN_WAYPOINTER: false
COACH_params:
 3
 4
 6
        stochastic: true
       FIXED_STEPS_PER_COM:
      checkin_frequency: 20
allow_agent_break: False
ACTION_PADDING: 0
10
        MIN_NEXT_ACTION_TIME: 19
MAX_NEXT_ACTION_TIME: 20
seed: 453413
12
13
14
        Agents:
         player_0:
              class_name: "SB_PPOWaypointActor"
16
            params:
                 policy_path:
```

8 examples/MAInspection/assets

8.1 examples/MAInspection/assets/typography.css

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

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

```
157
    .rc-slider-handle-4:hover {
158
159
        border-color: #ff006d;
160 }
161
162 /* ========= */
163
165 border-color: #adff02; 166 }
164 .rc-slider-handle-5 {
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
    .rc-slider-handle-6.rc-slider-handle-click-focused:focus {
183 border-color: #8f00ff; 184 }
182
185
187 border-color: #8f00ff; 188 }
    .rc-slider-handle-6:hover {
189
190 /* ========= */
```

9 examples/Multipolicy

9.1 examples/Multipolicy/Readme.md

```
# Multiploicy Director
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.
    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
    Finally, we construct a Dash app to display our solution.
    This example uses all most entirely default functionality. For more advanced functionality see the
         MAInspection example.
10
    <img src="Interface.png">
11
```

9.2 examples/Multipolicy/app.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
   \mbox{\#} Unless required by applicable law or agreed to in writing, software \mbox{\#} distributed under the License is distributed on an "AS IS" BASIS,
10
    # 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
   import sys
15
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
26
    import logging
    logging.basicConfig(level=logging.DEBUG, format="%(levelname)s:%(name)s:%(message)s")
   logger = logging.getLogger(__name__)
    pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
        name.startswith("pymunk")]
    for log_handler in pymunk_loggers:
        log_handler.setLevel(logging.INFO)
32
33
34
35
    def get_env():
36
        return PettingZooEnv(PZGame=waterworld_v4)
37
    if __name__ == "__main__":
38
        -- -- print("app - #################### RELOADING DASH APP #######################")
39
40
        MODEL_PATH = ""
41
        DENSE_PATH = min(glob.iglob(os.path.join(MODEL_PATH,'dense/*.zip')), key=os.path.getctime)
42
        EXPLORE_PATH = min(glob.iglob(os.path.join(MODEL_PATH,'explore/*.zip')), key=os.path.getctime)
43
44
45
        with open(os.path.join("Experiment", "train_director.yaml"), "r") as f:
46
                params = yaml.safe_load(f)
47
            except yaml.YAMLError as exc:
48
49
                print(exc)
50
51
        with open(os.path.join("Experiment","dash_app.yaml"), "r") as f:
52
53
                params["actor_params"] = yaml.safe_load(f)
54
            except yaml.YAMLError as exc:
55
56
                 print(exc)
57
        # external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
58
59
        env_factory = COACHIntegration(
60
            env_creator=get_env
61
            COACHEnvClass=COACHEnvironment,
            parameters=params
62
63
64
65
        ct.env_factory = env_factory
66
67
        locks = {
68
                     "action_card": False
69
                }
70
71
        proxy_url = re.sub("{{port}}", "8050", os.environ["VSCODE_PROXY_URI"])
72
        proxy_url = re.sub(os.environ["ACEHUB_BASEURL"], "", proxy_url)
        app = Dash(serve_locally=True, requests_pathname_prefix=proxy_url)
76
        app.layout = app_layout(env_factory)
```

78 app.run(debug=True)

9.3 examples/Multipolicy/train_agents.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
    # You may obtain a copy of the License at
6
         http://www.apache.org/licenses/LICENSE-2.0
   # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
   from __future__ import annotations
15
16
17
    from pettingzoo.sisl import waterworld_v4
    import supersuit as ss
18
19 from stable_baselines3 import PPO
    from stable_baselines3.ppo import MlpPolicy
    from stable_baselines3.common.callbacks import EvalCallback
22
   import time
24
    import glob
    from tqdm import tqdm
    import numpy as np
    import yaml
   import copy
30
   import torch
    CORES = 9
31
    torch.set_num_threads(CORES)
    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,
steps: int = 10_000,
43
        seed: int | None = 0,
44
45
        num_vec_envs=1,
46
        num_cpus=CORES.
47
        learning_rate=1e-3,
48
        batch_size=256,
        model dir="".
49
50
        name="",
        **env_kwargs
51
52
    ):
53
        model_path = os.path.join(model_dir, name)
54
        os.makedirs(model_path,exist_ok=True)
55
        # Train a single model to play as each agent in a cooperative Parallel environment
env = env_fn.parallel_env(**env_kwargs)
56
57
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,
             base_class="stable_baselines3")
64
65
        # Note: Waterworld's observation space is discrete (242,) so we use an MLP policy rather than CNN
66
        model = PPO(
67
            MlpPolicy,
68
             env,
69
70
             learning_rate=learning_rate,
             batch_size=batch_size,
71
72
             tensorboard_log=os.path.join("./tensorboard_log/", name)
        eval_env = env_fn.parallel_env(**env_kwargs)
        eval_env.reset(seed=seed)
76
        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,
              base_class="stable_baselines3")
 79
 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,
             "load_class": "SB_PPOPoliciesActor",
"env_params": env_kwargs
 87
 88
 89
         7
 90
         with open(weight_path + ".yaml", "w") as f:
 91
             yaml.dump(model_args, f, default_flow_style=False)
 92
 93
         model.learn(total_timesteps=steps, callback=eval_callback)
 94
 95
         model.save(weight_path)
 96
 97
98
         print("Model has been saved.")
         print(f"Finished training on {str(env.unwrapped.metadata['name'])}.")
99
100
101
         env.close()
102
103
         return weight_path
104
105
106
     def eval(
107
         env_fn,
108
         model_path,
109
         num_games: int = 10,
110
         render_mode: str | None = None,
111
         **env_kwargs
112
     ):
113
         # Evaluate a trained agent vs a random agent
114
         env = env_fn.env(render_mode=render_mode, **env_kwargs)
115
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
         # Note: We train using the Parallel API but evaluate using the AEC API
124
125
         # SB3 models are designed for single-agent settings, we get around this by using he same model for
             every agent
126
         for i in tqdm(range(num_games)):
127
             env.reset(seed=i)
128
             rewards = {agent: 0 for agent in env.possible_agents}
129
130
             for agent in env.agent_iter():
131
                  obs, reward, termination, truncation, info = env.last()
132
                  for a in env.agents:
133
                 rewards[a] += env.rewards[a] if termination or truncation:
134
135
136
                      for a in env.agents:
                          cum_rewards.append(rewards[a])
137
138
                      break
139
                  else:
140
                      act = model.predict(obs, deterministic=True)[0]
141
142
                 env.step(act)
         env.close()
143
144
         avg_reward = np.mean(cum_rewards)
145
         std_reward = np.std(cum_rewards)
146
147
         with open(model_path + ".txt", "a") as f:
148
             f.writelines(f"\n\n{env_kwargs}\n")
149
              f.writelines(f"\t Avg reward: {avg_reward}, std: {std_reward}\n")
150
151
             f.writelines(f"\t Rewards: {cum_rewards}\n")
152
         print(f"\t Avg reward: {avg_reward}, std: {std_reward}")
153
154
         return avg_reward
155
```

```
156
     if __name__ == "__main__":
    env_fn = waterworld_v4
157
158
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.
          model_path_dense = train_butterfly_supersuit(
165
              env_fn=env_fn,
steps=STEPS,
166
167
168
              seed=0,
              n_pursuers=1,
169
              n_evaders=10,
170
              n_poisons=40,
171
              model_dir="",
name="dense"
172
173
174
175
          ## Train Explore Eater
176
          model_path_explore = train_butterfly_supersuit(
177
178
              env_fn=env_fn,
              steps=STEPS,
179
180
              seed=0,
181
              n_pursuers=3,
182
              n_{evaders=5},
183
              n_poisons=5,
184
              model_dir="",
              name="explore")
185
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,
                  n_poisons=15,
206
207
```

9.4 examples/Multipolicy/train_director.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
\overset{.}{9} # Unless required by applicable law or agreed to in writing, software 10 # distributed under the License is distributed on an "AS IS" BASIS,
   # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
   from __future__ import annotations
   import sys
   sys.path.insert(0, "../../")
16
17
   import os
18
   import time
19
20
    import glob
    from tqdm import tqdm
22
    import numpy as np
23 from PIL import Image
24
   import yaml
^{25}
   from utilities.PZWrapper import PettingZooEnv
    from coach import COACHEnvironment
   from env import COACH_PettingZoo
30 from stable_baselines3 import PPO
31 from stable_baselines3.ppo import MlpPolicy
    from stable_baselines3.common.callbacks import EvalCallback
   import supersuit as ss
34
35
   from pettingzoo.sisl import waterworld_v4
36
37
    import torch
38
    CORES = 6
    torch.set_num_threads(CORES)
39
    torch.set_num_interop_threads(CORES)
40
41
42
    import logging
logging.basicConfig()
43
    logging.getLogger().setLevel(logging.INFO)
44
45
    logger = logging.getLogger(__name__)
46
    pymunk_loggers = [logging.getLogger(name) for name in logging.root.manager.loggerDict if
47
         name.startswith("pymunk")]
48
    for log_handler in pymunk_loggers:
        log_handler.setLevel(logging.INFO)
49
50
51
    # SB code adapted from https://pettingzoo.farama.org/tutorials/sb3/waterworld/
52
53
    ## We're going to train two policies, one that assumes a densely poisonous env
54
    ## and one that requires sparse exploration.
55
    # %%
56
    def train(
        env_fn,
57
58
        steps: int = 10_{-000},
59
        seed: int | None = 0,
60
        num_vec_envs=1,
61
        num_cpus=1,
62
        learning_rate=1e-3,
63
        batch_size=256,
64
        model_dir=""
65
   ):
66
        os.makedirs(model_dir,exist_ok=True)
67
68
        # Train a single model to play as each agent in a cooperative Parallel environment
69
        env = env_fn()
70
        env.reset(seed=seed)
71
        print(env.possible_agents)
72
        logger.info(f"Starting training on {str(env.metadata['name'])}.")
73
        env = ss.pettingzoo_env_to_vec_env_v1(env)
76
        env = ss.concat_vec_envs_v1(env, num_vec_envs=num_vec_envs, num_cpus=num_cpus,
             base_class="stable_baselines3")
```

```
77
                 # Note: Waterworld's observation space is discrete (242,) so we use an MLP policy rather than CNN
 78
                 model = PPO(
 79
 80
                       MlpPolicy,
 81
                        env.
 82
                        verbose=3.
 83
                        learning_rate=learning_rate.
 84
                        batch size=batch size.
                        tensorboard_log=os.path.join("./tensorboard_log/", model_dir.split("/")[-1])
 85
                )
 86
 87
 88
                 eval env = env fn()
 89
                 eval_env = ss.pettingzoo_env_to_vec_env_v1(eval_env)
                 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
                 model.learn(total_timesteps=steps, callback=eval_callback)
 93
 94
 95
                 \verb|model_name| = f"\{env.unwrapped.metadata.get('name')\}_{time.strftime('%Y%m%d-%H%M%S')\}" | fine.strftime('%Y%m%d-%H%M%S') | fine.strftime('%YmMd-%H%M%S') | fine.strftime('%YmMd-%H%M%S') | fine.strftime('%YmMd-%H%M%S') | fine.strftime('%YmMd-%H%M%S') | fine.strftime('%YmMd-%H%M%S') | fine.strftime('%YmMd-%H%M%S') | fine.strftime('%YmMd-%H%MM%S') | fine.strftime('%YmMd-%H%MM%S') | fine.strftime('%YmMd-%H%MMS') | fine.strftime('%MMMS') | fine.strftime('%MMS') | fine.strftime('%MMMS') | fine.strftime('%MMS') | fine.strftime('%MMMS') | fine.strftime('%MMS') | fine.strftime('%MMS') | fine.strftime('%MMS') | fine.strftime('%MMS') | fine.strf
                 model_path = os.path.join(model_dir, model_name)
 96
 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,
                 num_games: int = 10,
109
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(
                        f"\nStarting evaluation on {str(env.metadata['name'])} (num_games={num_games}, render={render})"
117
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 he 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}
                        running = True
133
134
                        i = 0
135
                        while running:
136
                                j += 1
                                act = {"director": model.predict(obs["director"], deterministic=False)[0]}
137
138
                               logger.debug("Obs From Director: %s", obs["director"]) # DEBUG
logger.debug("Action From Director: %s", act) # DEBUG
139
140
141
142
                                obs, reward, term, trunc, info = env.step(act)
143
                                for a in env.agents:
                                       rewards[a] += reward[a]
144
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.trajectory.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
```

```
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
159
          env.close()
160
          avg_reward = np.mean(cum_rewards)
std_reward = np.std(cum_rewards)
161
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
170
          return avg_reward
171
172
173
     def get_env():
174
          return PettingZooEnv(PZGame=waterworld_v4)
175
     176
177
          DENSE_PATH = min(glob.iglob(os.path.join(MODEL_PATH,'dense/*.zip')), key=os.path.getctime)
178
          EXPLORE_PATH = min(glob.iglob(os.path.join(MODEL_PATH,'explore/*.zip')), key=os.path.getctime)
179
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
          for agent, param in params["COACH_params"]["Agents"].items():
    param["params"]["policy_paths"]["dense"] = DENSE_PATH
    param["params"]["policy_paths"]["explore"] = EXPLORE_PATH
187
188
189
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)
               env.augment(params)
195
196
               env.reset()
197
               return env
198
199
          model_path = train(
200
               get_env_pz,
201
               steps=100,
               seed=0,
202
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
          with open(os.path.join("Experiment", "dash_app_template.yaml"), 'r') as f:
    with open(os.path.join("Experiment", "dash_app.yaml"), 'w') as g:
214
215
216
                    for line in f.readlines():
                        line = line.replace("<DENSE_PATH>", DENSE_PATH)
217
                        line = line.replace("<EXPLORE_PATH>", EXPLORE_PATH)
line = line.replace("<DIRECTOR_PATH>", model_path)
218
219
                        g.write(line)
220
221
222 # %%
```

10 examples/Multipolicy/Experiment

10.1 examples/Multipolicy/Experiment/dash_app.yaml

```
2
      Default: # Name of Interface
3
        interface_class: DefaultInterface
        models:
            class_name: DefaultActor
      Random: # Name of Interface
        interface_class: RandomActionInterface
10
        models:
11
          Random:
12
            class_name: RandomActor
13
      Dense_v_Sparce: # Name of Interface
14
15
        interface_class: SBPolicyInterface
        iterface_parameters:
16
17
          n_policies: 2
18
         max_action_len: 10
19
        roles: [pursuer_0, pursuer_1, pursuer_2] # If this is not supplied, assumed all roles
20
21
        models:
22
          Dense_v_Sparse_v1:
23
            class_name: SB_PPOPoliciesActor
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:
            class_name: SB_PPOPoliciesActor
31
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:
        interface_class: SBPolicyInterface
39
40
        iterface_parameters:
41
          n_policies: 2
42
          max_action_len: 10
43
44
        models:
45
          Tall_v_Short:
46
            class_name: SB_PPOPoliciesActor
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
55
     PPO_Director:
56
        class_name: SB3_PPO_Director
57
        path: director/waterworld_v4_20240213-190312.zip
58
         pursuer_0:
            classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
          pursuer_1:
            classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
63
          pursuer_2:
            classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
```

10.2 examples/Multipolicy/Experiment/dash_app_template.yaml

```
interfaces:
   Default: # Name of Interface
1
2
3
        \verb|interface_class: DefaultInterface|\\
4
        models:
5
          Default:
6
            class_name: DefaultActor
8
      Random: # Name of Interface
9
       \verb|interface_class|: RandomActionInterface|
10
        models:
11
          Random:
12
            class_name: RandomActor
13
14
      Dense_v_Sparce: # Name of Interface
       interface_class: SBPolicyInterface
16
        iterface_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
^{22}
            class_name: SB_PPOPoliciesActor
            params:
              policy_paths:
                dense: <DENSE_PATH>
                explore: <EXPLORE_PATH>
              max_action_len: 10
30
          Dense_v_Sparse_v2:
            class_name: SB_PPOPoliciesActor
            params:
33
              policy_paths:
                dense: <DENSE_PATH>
34
35
                explore: <EXPLORE_PATH>
36
              max_action_len: 10
37
38
      Tall_v_Short:
       interface_class: SBPolicyInterface
39
40
        iterface_parameters:
         n_policies: 2
41
42
          max_action_len: 10
43
44
        models:
45
          Tall v Short:
            class_name: SB_PPOPoliciesActor
46
47
            params:
48
              policy_paths:
                dense: <DENSE_PATH>
49
                explore: <EXPLORE_PATH>
50
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
      PPO_Director:
        class_name: SB3_PP0_Director
56
57
        path: <DIRECTOR_PATH>
58
        roles:
         pursuer_0:
59
60
            classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
61
          pursuer_1:
62
            classes: [Dense_v_Sparse_v1, Dense_v_Sparse_v2]
63
            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 r_pursuers: 3
```

10.4 examples/Multipolicy/Experiment/train_director.yaml

```
env_params:
2
      n_pursuers: 3
3
      max_cycles: 30
    COACH_params:
4
      stochastic: true
6
      FIXED_STEPS_PER_COM:
      checkin_frequency: 10
      allow_agent_break: False ACTION_PADDING: 0
9
10
      MIN_NEXT_ACTION_TIME: 1
      MAX_NEXT_ACTION_TIME: 10
11
12
      seed: 453413
13
      Agents:
14
       pursuer_0:
         class_name: "SB_PPOPoliciesActor"
16
17
            max_action_len: 10
18
            policy_paths:
              dense:
20
              explore:
       pursuer_1:
22
          class_name: "SB_PPOPoliciesActor"
         params:
            max_action_len: 10
            policy_paths:
              dense:
              explore:
      pursuer_2:
         class_name: "SB_PPOPoliciesActor"
         params:
30
            max_action_len: 10
            policy_paths:
dense:
34
              explore:
```

11 examples/Multipolicy/assets

$11.1 \quad examples/Multipolicy/assets/typography.css$

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

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

```
157
    .rc-slider-handle-4:hover {
158
159
        border-color: #ff006d;
160 }
161
162 /* ========= */
163
165 border-color: #adff02; 166 }
164 .rc-slider-handle-5 {
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
    .rc-slider-handle-6.rc-slider-handle-click-focused:focus {
183 border-color: #8f00ff; 184 }
182
185
187 border-color: #8f00ff; 188 }
    .rc-slider-handle-6:hover {
189
190 /* ========= */
```

12 utilities

12.1 utilities/PZParams.py

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

12.2 utilities/PZWrapper.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
   # You may obtain a copy of the License at
        http://www.apache.org/licenses/LICENSE-2.0
   \mbox{\#} Unless required by applicable law or agreed to in writing, software \mbox{\#} distributed under the License is distributed on an "AS IS" BASIS,
10
   # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
11
12
   # See the License for the specific language governing permissions and
   # limitations under the License.
14
   # The following code is modified from Farama-Foundation/PettingZoo
15
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
   from collections.abc import Callable # for type hinting from typing import Any, Union, cast # for type hinting
24
25
   from gymnasium.wrappers import FlattenObservation
    from pettingzoo.classic import connect_four_v3 as PZGame # type: ignore[import]
   from pettingzoo.sisl import waterworld_v4 as PZGame
28
   logging.getLogger("pettingzoo.utils.env_logger").setLevel(logging.WARNING)
   logger = logging.getLogger(__name__)
33
   import copy
34
35
   import numpy as np
36
   from gymnasium.spaces import Box, Discrete, flatten, flatten_space, unflatten
    from numpy.typing import NDArray
38
   from pettingzoo.utils.conversions import aec_to_parallel
39
   from pettingzoo.utils.env import AECEnv, ParallelEnv
40
   from .PZParams import PettingZooEnvParams
41
42
43
   44
45
   ## Auxiliary Functions
46
   def softmax(x):
47
        exp = np.exp(x)
48
        return exp / cast(float, np.exp(x).sum())
49
50
51
52
    Qunflatten.register(Discrete)
53
   def _unflatten_discrete(
        space: Discrete, x: Union[NDArray[np.int64], NDArray[float]]
54
   ) -> np.int64:
55
56
        nonzero = np.nonzero(x)
        if len(nonzero[0]) == 0:
57
           raise ValueError(
58
               f''\{x\} is not a valid one-hot encoded vector and can not be unflattened to space \{space\}.
59
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]:
         ""Returns the class to use, based on input arguments
72
73
74
        args: argparse.Namespace
           arguments that were passed to the 'main()' function
```

```
79
        Returns
 80
 81
         class
         the class to use in creating env objects
 82
 83
 84
         return PettingZooEnv
 85
 86
    \#\# This is what you want to edit if you add wrappers, or change
 87
 88
    ## things about the env parameters.
     def env_creator(**kwargs):
 89
 90
         return PZGame.env(**kwargs)
 91
 92
    93
    ## Main Class
 94
     95
 96
    class PettingZooEnv:
 97
        def __init__(self, PZGame) -> None:
98
99
            def env_creator(**kwargs):
100
                return PZGame.parallel_env(**kwargs)
101
             self.env_creator = env_creator
102
            # If you have wrappers define a function to set it up properly
self.standard_params = {"render_mode": "rgb_array"}
103
104
105
             self.current_params = None
106
107
         def __getattr__(self, name):
              "<sup>"</sup>Returns an attribute with ''name'', unless ''name'' starts with an underscore."""
108
             if name == "env":
109
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):
             if action is not None:
117
118
                return unflatten(action_space, action)
119
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
                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
            # logger.info(f"action: {str(obs)}, {self.env.terminations}, {self.env.truncations}")
136
137
            return (
138
                obs.
139
                self.env.rewards.
140
                self.env.terminations.
141
                self.env.truncations.
142
                self.env.infos,
143
144
         def reset(self):
145
146
            if self.parallel:
147
                return self.env.reset()
148
            self.env.reset()
149
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)
                if isinstance(self.env, ParallelEnv):
self.parallel = True
165
166
                else:
167
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()
                for agt in self.env.possible_agents:
    self.action_spaces[agt] = flatten_space(self.env.action_space(agt))
    self.observation_spaces[agt] = flatten_space(
176
177
178
179
                          self.env.observation_space(agt)
180
                     self.action_type[agt] = type(self.env.action_space(agt))
181
182
           def seed(self, seed):
183
                if hasattr(self.env, "seed"):
184
185
                     self.env.seed(seed)
186
           def render(self, *args, **kwargs):
    if kwargs.get("close", False):
187
188
                     self.env.close()
189
190
                     return
191
192
                return self.env.render()
```

12.3 utilities/iotools.py

```
# Copyright (c) 2023 Mobius Logic, Inc.
1
3
   # Licensed under the Apache License, Version 2.0 (the "License");
4
   # you may not use this file except in compliance with the License.
   # You may obtain a copy of the License at
        http://www.apache.org/licenses/LICENSE-2.0
   \mbox{\#} Unless required by applicable law or agreed to in writing, software \mbox{\#} distributed under the License is distributed on an "AS IS" BASIS,
10
   # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   # See the License for the specific language governing permissions and
   # limitations under the License.
14
15
   """Contains various utility functions/classes used in the project"""
16
17
   import csv
   import json
18
19
   import os
20
   from collections.abc import Callable # for type hinting
   from typing import Any, Union, cast, NewType # for type hinting
22
   import numpy as np
24
25
   EnvId = NewType("EnvId", str)
26
   PathString = str
   ## Auxiliary Functions
   ### Turn dictionary with keyed tuple into csv
   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
       dct : dict
43
44
           Dict of values with keys given as tuples
45
        filename : str
46
           Name of file to save to
47
        do_sort : boolean, optional
            whether to sort the output keys
48
           This is assuming they are of the form "Env_#"
49
50
       Side-Effects
51
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,
              (c,a): val_ca, (c,b): val_cb,
(d,a): val_da, (d,b): val_db}
64
65
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
        rectangular matrix.
        \# split (1, r) keys into lists of 1 and r
        1_keys = list({first for first, second in dct})
       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
                 with open(filename, mode="w", newline="", encoding="utf8") as file:
 85
 86
                         csvfile = csv.writer(
                                file, delimiter=",", quoting=csv.QUOTE_MINIMAL, lineterminator="\n"
 87
 88
                        header = [""] + r_keys # [""] to add blank entry to csv row
 89
                        csvfile.writerow(header)
 90
 91
                        for first in l_keys:
    row = [first] + [dct.get((first, second), "") for second in r_keys]
 92
 93
 94
                                csvfile.writerow(row)
 95
 96
 97
         def load_keyed_tuple(
 98
                 filename: PathString,
                 format_function: Callable[[str], Any] = lambda x: x,
 99
100
         ) -> dict[tuple[EnvId, EnvId], Any]:
101
                 Read a tuple-indexed dict (a matrix) from a csv file.
102
103
104
                 This is the reverse of save_keyed_tuple
                 The format_function is used to convert the values from string to whatever format is desirable. The default leaves it as a string.
105
106
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
117
118
                 dict
                       The dict that was read
119
120
121
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,
                              (c,a): val_ca, (c,b): val_cb,
(d,a): val_da, (d,b): val_db}
132
133
134
135
                 dct: dict[tuple[EnvId, EnvId], Any] = {}
                 with open(filename, mode="r", newline="", encoding="utf8") as file:
    csvfile = csv.reader(file, delimiter=",")
136
137
138
                        \mbox{\tt\#} first line has an empty spot to account for alignment
                        # of columns. So, we ignore that
139
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
                                        l_key = cast(EnvId, l_key)
r_key = cast(EnvId, r_key)
dct[(l_key, r_key)] = format_function(value)
144
145
146
147
                 return dct
148
149
151
         ## Numpy Encoders for JSON
         152
153
         \hbox{\tt\#\# Recursively encodes objects with a reprJSON function}
154
          \verb|# https://stackoverflow.com/questions/5160077/encoding-nested-python-object-in-json/com/stackoverflow.com/questions/5160077/encoding-nested-python-object-in-json/com/stackoverflow.com/questions/5160077/encoding-nested-python-object-in-json/com/stackoverflow.com/questions/5160077/encoding-nested-python-object-in-json/com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackoverflow.com/stackover
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
    # with open(filename, 'w') as jsonfile:
163
    # json.dump(edge, jsonfile, cls=NumpyEncoder)
# with open(filename, 'r') as jsonfile:
164 #
165
           edge1 = json.load(jsonfile, cls=NumpyDecoder)
166
167
168
     class NumpyEncoder(json.JSONEncoder):
169
         """Encode numpy data for JSON writer"""
170
171
         {\tt def \ default(self, \ o):} \quad {\tt \# \ type: \ ignore \ \ \# \ this \ is \ called \ by \ the \ JSON \ library
172
173
             if isinstance(o, np.integer):
                 return {"np.integer": int(o)}
174
             if isinstance(o, np.floating):
175
176
                 return {"np.floating": float(o)}
             if isinstance(o, np.ndarray):
177
178
             return {"np.array": o.tolist()}
return json.JSONEncoder.default(self, o)
179
180
181
     class NumpyDecoder(json.JSONDecoder):
182
183
         """Decode numpy data from JSON""
184
         def __init__(self, *args, **kwargs): # type: ignore # this is called by the JSON library
185
186
             json.JSONDecoder.__init__(self, object_hook=self.numpy_hook, *args, **kwargs)
187
         def numpy_hook(self, dct): # type: ignore # this is called by the JSON library
    """Convert dict with numpy data to numpy object"""
188
189
             if "np.integer" in dct:
190
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
200 ## TensorFlow Logging
201
     202
     class TBWriter:
         def __init__(self, args: dict[str, Any]) -> None:
203
204
205
             import time
206
207
             from tensorflow import summary # type: ignore[import]
208
209
             self.summary = summary
             log_dir = args["log_dir"]
210
211
212
             now = time.localtime()
213
             subdir = time.strftime("d-\%b-\%Y_{M}.\%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:
             new_dir = os.path.join(self.summary_dir, name)
219
220
             self.summary_writer[name] = self.summary.create_file_writer(new_dir)
221
         def write_item(self, name: str, label: str, data: Any, step: int) -> None:
222
             if label not in self.summary_writer.keys():
223
224
                 self.create_scalar_writer(label)
225
             with self.summary_writer[label].as_default():
226
227
                 if isinstance(data, (np.ndarray, np.generic)):
228
                     data = data.item()
229
                 if hasattr(data, "__len__"):
230
                     data = data[0]
231
232
                 self.summary.scalar(name=name, data=data, step=step)
233
234
             self.summary_writer[label].flush()
235
236
         def write_items(self, name: str, data: dict[str, list[float]], step: int) -> None:
             for label in data.keys():
237
                 self.write_item(name, label, data[label], step)
238
239
```

240

```
class Telemetry:

def __init__(self) -> None:
    self.loggers: list[TBWriter] = []

def add_logger(self, logger: TBWriter) -> None:
    self.loggers.append(logger)

def write_item(self, name: str, label: str, data: Any, step: int) -> None:
    for logr in self.loggers:
        logr.write_item(name, label, data, step)

def write_items(self, name: str, data: dict[str, list[float]], step: int) -> None:
    for logr in self.loggers:
    logr.write_items(name, data, step)
```

12.4 utilities/planning.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
3
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
4
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
  # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
    from .timelines import Timeline, Timelines, TimelineEvent
16
    import numpy as np
17
    from numpy.random import Generator, PCG64DXSM
    import copy
18
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
            self.cummulative_rews = dict()
        def __repr__(self):
30
            return self.__str__()
        def __str__(self):
            return f"parameters: {self.parameters}, seed: {self.seed}, trajectory len:
                 {len(self.trajectory)}'
35
    class Trajectory:
37
        def __init__(self, env, initial_return, intial_frame=None):
             self.players = copy.copy(env.possible_agents)
38
39
             self.trajectory = []
            self.actions = []
40
41
            self.step_returns = []
            self.agent_info = []
42
43
             self.add(
44
                env, None, initial_return, None
45
46
             self.frames = []
            if intial_frame is not None:
47
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,
                     "step_return": step_return,
"agent_info": agent_info
54
55
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
72
    ## Telemetry Object:
    class Telemetry:
        def __init__(self, name, **kwargs):
          self.data = None
76
            self.data_labels = None
            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:
              shape = getattr(self.data, "shape", None)
return f"Telemetry - name: {self.name}, shape: {shape}"
 91
 92
 93
 94
          def \__repr\__(self) \rightarrow str:
 95
              return self.__str__()
 96
 97
          def set_data(self, data):
 98
              self.data = data
              self.data_labels = [""] * data.shape[0]
99
              self.colors = [None] * data.shape[0]
100
101
102
          def as_df(self):
              # print("Telemtry:", self.data, self.data_labels, self.xscale)
103
104
105
              tmp = pd.DataFrame(self.data.T)
106
107
              # print("Telemtry:", tmp)
              columns = copy.copy(self.data_labels)
108
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("Telemtry:", tmp)
119
              tmp.index = self.xscale
              return tmp
120
121
122
     # Returns a fixed seed and fixed set of parameters
123
     class BaseParameterGenerator:
124
          def __init__(self, seed, param):
125
              self.seed = seed
              self.param = param
126
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
     # This returns a random seed but a fixed set of parameters
143
144
     class SeededParameterGenerator:
145
          \label{def_lemma} {\tt def} \ \_{\tt init}\_{\tt (self, seed, param)}:
              self.np_random = Generator(PCG64DXSM(seed=seed))
146
              self.param = param
self._frozen = False
147
148
149
150
          def sample(self):
151
              if self._frozen:
                  return (self._frozen_random, self.param)
152
153
154
                  return (self.np_random.integers(low=2**32), self.param)
155
          def setseed(self, seed):
156
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):
          @staticmethod
171
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 secretely overload the standard timeline behavior to make it
181
               ## infinite
182
               tmp = CommunicationSchedule(length=1, allow_agent_break=allow_agent_break)
              tmp.__dict__["checkin_frequency"] = checkin_frequency
tmp.__dict__["checkin_offset"] = checkin_offset
183
184
               tmp.__dict__["blackout_frequency"] = blackout_frequency
185
              tmp.__dict__["blackout_length"] = blackout_length
tmp.__dict__["blackout_offset"] = blackout_offset
186
187
              tmp.__dict__["allow_agent_break"] = allow_agent_break,
tmp.__dict__["repeating"] = True
188
189
190
191
               blackout_event = TimelineEvent(parameters=None, label="blackouts")
               checkin_event = TimelineEvent(parameters=None, label="checkins")
192
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:</pre>
204
                        return True
205
206
                   return False
207
208
              tmp.blackouts.__dict__["get"] = get
209
210
              return tmp
211
212
213
          @staticmethod
          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")
checkin_event = TimelineEvent(parameters=None, label="checkins")
216
217
218
219
              for t in blackouts:
220
                   tmp.blackouts.add_events(t, blackout_event)
221
222
              for t in checkins:
223
                   tmp.checkins.add_events(t, checkin_event)
224
225
              return tmp
226
          def __init__(self, length, allow_agent_break=True):
227
               super().__init__(labels=["blackouts", "checkins"], length=length)
self.ALLOW_AGENT_BREAK = allow_agent_break
228
229
230
               self.repeating = False
231
232
              def get(time, role):
233
                   return False
234
235
               self.blackouts.__dict__["get"] = get
236
               # self.checkins.add_event(time=0, event=TimelineEvent(parameters=None, label="checkin"))
237
238
          def __getstate__(self):
239
               return self.__dict__
```

```
240
241
            def __setstate__(self, state):
242
                  self.__dict__ = state
243
244
            def __deepcopy__(self, memo):
    if self.repeating:
245
246
                      return CommunicationSchedule.repeating(
247
                            checkin_frequency=copy.copy(self.checkin_frequency),
checkin_offset=copy.copy(self.checkin_offset),
blackout_frequency=copy.copy(self.blackout_frequency),
blackout_length=copy.copy(self.blackout_length),
blackout_offset=copy.copy(self.blackout_offset),
248
249
250
251
252
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
      class COA(Timeline):
261
262
            def to_dict(self):
263
                 d = {
                       time: {event.label: event.to_dict() for event in events} for time, events in
264
                             self.timeline.items()
                       }
265
266
                  return d
267
268
            def get_actions(self, t):
^{269}
                  return self.timeline.get(t, default=None)
```

12.5 utilities/timelines.py

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
3
    # Licensed under the Apache License, Version 2.0 (the "License");
4
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
         http://www.apache.org/licenses/LICENSE-2.0
  # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   # 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
24
    # Note: Everything here has been written so that it (at present)
    # deepcopies. If you do anything to make a model not deepcopy
   # by default you'll need to add a __deepcopy__ function
    #######################
    class TimelineEvent:
        id_idx = 0
        @staticmethod
33
        def from_dict(event):
            tmp = TimelineEvent(
34
35
                 event["label"],
36
                 event["parameters"],
                 tags = {k:v for k,v in event.items() if (k not in {"label", "parameters", "id"})},
37
                 id = event.get("id", None)
38
39
40
41
            return tmp
42
        def __init__(self, label, parameters, tags: dict = dict(), id=None):
43
44
            if id == None:
                 self.id = TimelineEvent.id_idx
45
46
                 TimelineEvent.id_idx += 1
47
48
             self.label = label
49
             self.parameters = parameters
            self.tags = tags
50
51
52
        def to_dict(self):
53
            return {
                 "label": self.label,
54
                 "parameters": self.parameters,
"tags": self.tags
55
56
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)
    class Timeline:
       def __init__(self, length=0, fixedlength=None):
            self.timeline = dict()
```

```
79
             self.events = dict()
             self.labels = dict()
 80
 81
 82
             self.fixedlength = False
             self.length = length
 83
 84
             self.allow_dup_labels = False
 85
 86
             if fixedlength is not None:
 87
                 if length >0:
                     raise Exception("Both length and fixedlength are specified. Only one should be.")
 88
 89
                 self.fixedlength = True
 90
                 self.length = fixedlength
 91
         def __repr__(self) -> str:
 92
             return f"Timeline: {str(self.events)}"
 93
 94
 95
         {\tt 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 = {
                   "start": int,
104
             #
                   "action": str,
105
             #
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
118
                 for t, e in events:
                     self.add_event(t, e)
119
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
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:
                      self.delete_event_at(time=time, event=e_del)
135
136
137
138
             if time not in self.timeline.keys():
                 self.timeline[time] = set()
139
140
             if event not in self.events.kevs():
141
142
                 self.events[event] = set()
143
             if event.label not in self.labels.keys():
144
145
                 self.labels[event.label] = set()
146
             self.timeline[time].add(event)
147
             self.timeline = dict(sorted(self.timeline.items()))
148
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]
                  if time == self.length-1 and not self.fixedlength:
161
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():
                  t = list(self.events[event])[0]
170
171
                  self.delete_event_at(event=event, time=t)
172
         def move_events(self, time, to, events=None):
173
174
              if events is None:
                  events = copy.copy(self.timeline[time]) # we're modifying the object as we iterate through it
175
176
177
             for event in events:
                  self.delete_event_at(event = event, time=time)
self.add_event(event=event, time=to)
178
179
180
181
182
         def add_interval(self, event: TimelineInterval):
183
184
              for t in event.range:
185
                  self.add_event(t, event)
186
         def first(self, event):
187
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
199
200
              return t + 1
201
         def interval_end(self, event, t):
202
203
              if t not in self.events[event]:
204
                 return None
205
206
              while t in self.events[event]:
207
208
209
             return t - 1
210
211
         def get_all_labeled(self, label):
              return self.labels[label]
212
213
214
         def get_events_at_time(self, time):
215
              return self.timeline[time]
216
217
         def get(self, time, label):
218
              r = []
              if label not in self.labels.keys():
    raise Exception(f"'{label}' not found in known labels: {list(labels.keys())}")
219
220
221
             for t, event in self.labels[label]:
222
223
                  if t == time:
                      r.append(event)
224
225
226
             if len(r) == 1:
227
                  return r[0]
228
229
             if len(r) == 0:
230
                  return None
231
              if len(r) > 1:
232
                  raise Exception(f"Multiple events with same label at same time: {r}")
233
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):
             L = list(self.timeline.keys())
247
248
              if len(L) == 0:
249
                  return None, None
250
             last_high = L[-1]
if time >= last_high:
251
252
                  return None, None
253
              if time < L[0]:
254
255
                  return L[0]. self.timeline[L[0]]
256
257
              # Go through timeline vis bisection. Probably overkill
258
              while len(L)>1:
259
                  half_idx = int(len(L)/2)
260
                  if L[half_idx] > time:
261
                      last_high = L[half_idx]
L = L[:half_idx]
262
263
264
                  else:
265
                      L = L[half_idx:]
266
                      if L[-1] > time:
                          last_high = L[-1]
267
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
              LABELLENGTH = 8
289
290
              TLSPACE = LINEWIDTH - PADDING - LABELLENGTH
291
292
              if self.length < TLSPACE:
293
                  step = 1
294
                  LINEWIDTH = LABELLENGTH + self.length
295
                  TLLENGTH = self.length
296
              else:
297
                  step = int(np.ceil(self.length/(TLSPACE)))
298
                  TLLENGTH = int(np.ceil(self.length/step))
299
300
              ## Time:
301
              ticklabs = [" "]*LINEWIDTH
              ticklabs[0:5] = list("Time:")
302
              ticks = [" "]*LABELLENGTH + ["-"]*(TLLENGTH+1)
303
304
305
              demarks = list(range(0,self.length, step*10))
306
              for j, i in enumerate(demarks):
307
308
                  j = LABELLENGTH + j*10
309
                  c = len(str(i))
tmp = [" "]*7
if c < 8:</pre>
310
311
312
                      tmp[(3-int((c-1)/2)):(3-int((c-1)/2)) + c] = str(i)
313
314
                  else:
                      tmp = "{:.2E}".format(c)
315
316
317
                  ticklabs[j-3:j+4] = tmp
318
                  ticks[j] ="|"
319
320
             ticks[-1] ="|"
321
```

```
322
323
              ## Lines
324
325
              lines = {
                  label: list(str(label)[:LABELLENGTH]) + [" "] * max(0,LABELLENGTH - len(label)) + [" "]*TLLENGTH
326
327
                   for label in self.labels.keys()
328
329
              for label, events in self.labels.items():
330
331
                  for t, event in events:
    lines[label][LABELLENGTH + int(t/step)] = "o"
332
333
              lines = ["".join(line) for line in lines.values()]
lines.insert(0, "".join(ticklabs))
lines.insert(1, "".join(ticks))
334
335
336
337
              return "\n".join(lines)
338
339
340
341
     class Timelines:
          def __init__(self, labels, length=0, fixedlength = None):
342
              self.timelines = {label:Timeline(length=length, fixedlength=fixedlength) for label in labels}
343
344
345
          def get_events(self, time):
              return {label: tl.get_events_at_time(time) for label, tl in self.timeslines.items()}
346
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):
                 tmp = self.__class__(labels=[])
355
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

```
# Copyright (c) 2024 Mobius Logic, Inc.
1
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.
    # You may obtain a copy of the License at
6
         http://www.apache.org/licenses/LICENSE-2.0
8
    \mbox{\#} Unless required by applicable law or agreed to in writing, software \mbox{\#} distributed under the License is distributed on an "AS IS" BASIS,
10
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12
    # See the License for the specific language governing permissions and
    # limitations under the License.
14
15
    from typing import Any
16
17
18
    class MultiDict:
19
         def __init__(self, tags: dict = dict()):
             self.tags = tags # tags - str(type): list(values)
self._vals = dict()
20
21
22
             self._refs = {tag: {val:[] for val in values} for tag, values in tags.items()}
23
24
        def __getattr__(self, __name: str) -> Any:
    if __name in self.tags.keys():
25
26
                 return SliceMakerDict(__name, self)
        def __getitem__(self, select_name: dict):
    new_tags = dict(self.tags)
28
29
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])
                 for tag_type_name in select_name.keys():
41
42
                      del new_tag[tag_type_name]
43
44
                 new_mdct[new_tag] = item
45
46
             return new mdct
47
48
         def __setitem__(self, key: dict, value: Any):
49
50
             # Following are sufficient to check all keys supplied:
             if not len(key.keys()) == len(self.tags.keys()):
51
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
                           {self.tags.keys()}")
59
60
                 if tag not in tags:
61
                      self.tags[tag_type].append(tag)
62
                      tags[tag] = list()
63
64
                 tags[tag].append(value)
65
66
             self._vals[value] = key
67
68
         def __repr__(self) -> str:
69
70
             tab_count = []
             for tag_type in self.tags.keys():
71
72
                 count = int(np.floor(len(tag_type) / 8)) + 1
                 tab_count.append(count)
                 s += f"{tag_type}\t'
76
             s += "value: \n"
```

```
78
              for item, tag in self._vals.items():
 79
 80
                   # Keep the same order as above
                   for i, tag_type in enumerate(self.tags.keys()):
 81
 82
                       max_len = tab_count[i]*8 - 2
                       tag_len = len(str(tag[tag_type]))
 83
 84
                       if tag_len > max_len:
 85
                            sp = str(tag[tag_type])[tab_count[i]*8-2] + "\t"
 86
 87
                            count = int(np.floor((max_len - tag_len)/8) + 1)
sp = str(tag[tag_type]) + "\t"*count
 88
 89
 90
                       s += sp
 91
 92
                   s += str(item)
 93
                   s += "\n"
 94
 95
 96
              return s
97
98
     class SliceMakerDict:
          def __init__(self, tag_type_name, parent: MultiDict):
99
100
              self.parent: MultiDict = parent
101
              self.tag_type_name = tag_type_name
102
          def __repr__(self) -> str:
103
104
              return str(self.parent.tags[self.tag_type_name])
105
          def __getitem__(self, key):
    new_tags = dict(self.parent.tags )
106
107
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
     def unit_vector(vector):
          """Returns the unit vector of the vector."""
125
          return vector / np.linalg.norm(vector)
126
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
          >>> angle_between((1, 0, 0), (1, 0, 0))
134
          0.0
135
          >>> angle_between((1, 0, 0), (-1, 0, 0))
136
137
          3.141592653589793
138
139
          v1_u = unit_vector(v1)
          v2_u = unit_vector(v2)
140
          \texttt{return np.arccos(np.clip(np.dot(v1\_u, v2\_u), -1.0, 1.0))}
141
142
143
144
     \label{lem:def_plot_traj} \mbox{def plot_traj(p, q, v, traj=None, w=[0, 0], ax=None):}
145
          if ax is None:
146
              ax = plt.gca()
          if not traj is None:
    ax.plot(traj[:, 0], traj[:, 1], "--")
147
148
          ax.plot(p[0], p[1], "or")
ax.plot(q[0], q[1], "og")
149
150
          ax.quiver(p[0], p[1], v[0], v[1], angles="xy", scale_units="xy", scale=1)
151
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
       def plot_traj_3d(p, q, v, traj=None, w=[0, 0], ax=None):
    if ax is None:
160
161
162
                   ax = plt.figure().add_subplot(projection="3d")
             ax = plt.figure().add_subplot(projection="3d")
if not traj is None:
    ax.plot(traj[:, 0], traj[:, 1], traj[:, 2], "--")
ax.plot(p[0], p[1], p[2], "or")
ax.plot(q[0], q[1], q[2], "og")
ax.quiver(p[0], p[1], p[2], v[0], v[1], v[2])
ax.quiver(p[0], p[1], p[2], float(w[0]), float(w[1]), float(w[2]))
return ax
163
164
165
166
167
168
169
              return ax
170
171
       def r(t, w, p, v):
    return p + (t**2) * w / 2 + t * v
172
173
174
175
       def r_p(t, w, v):
    return t * w + v
176
177
178
179
       def g(t, w, t1, p, v):
    if t < t1:</pre>
180
181
                  return r(t, w, p, v)
182
183
                    return r(t1, w, p, v) + (t - t1) * r_p(t1, w, v)
184
185
186
       def dg(t, w, t1, p):
    return r(t1, w, p) + t * r_p(t1, w, p)
187
188
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
```