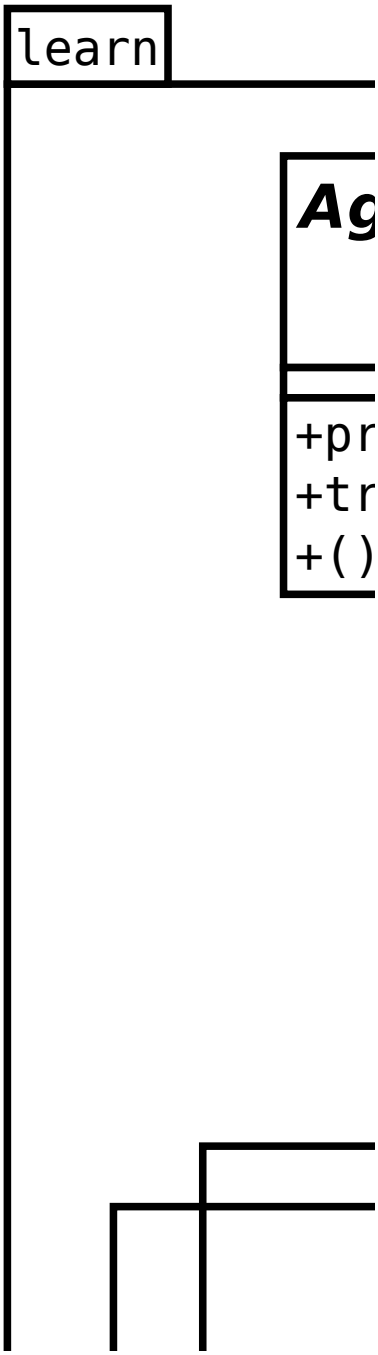
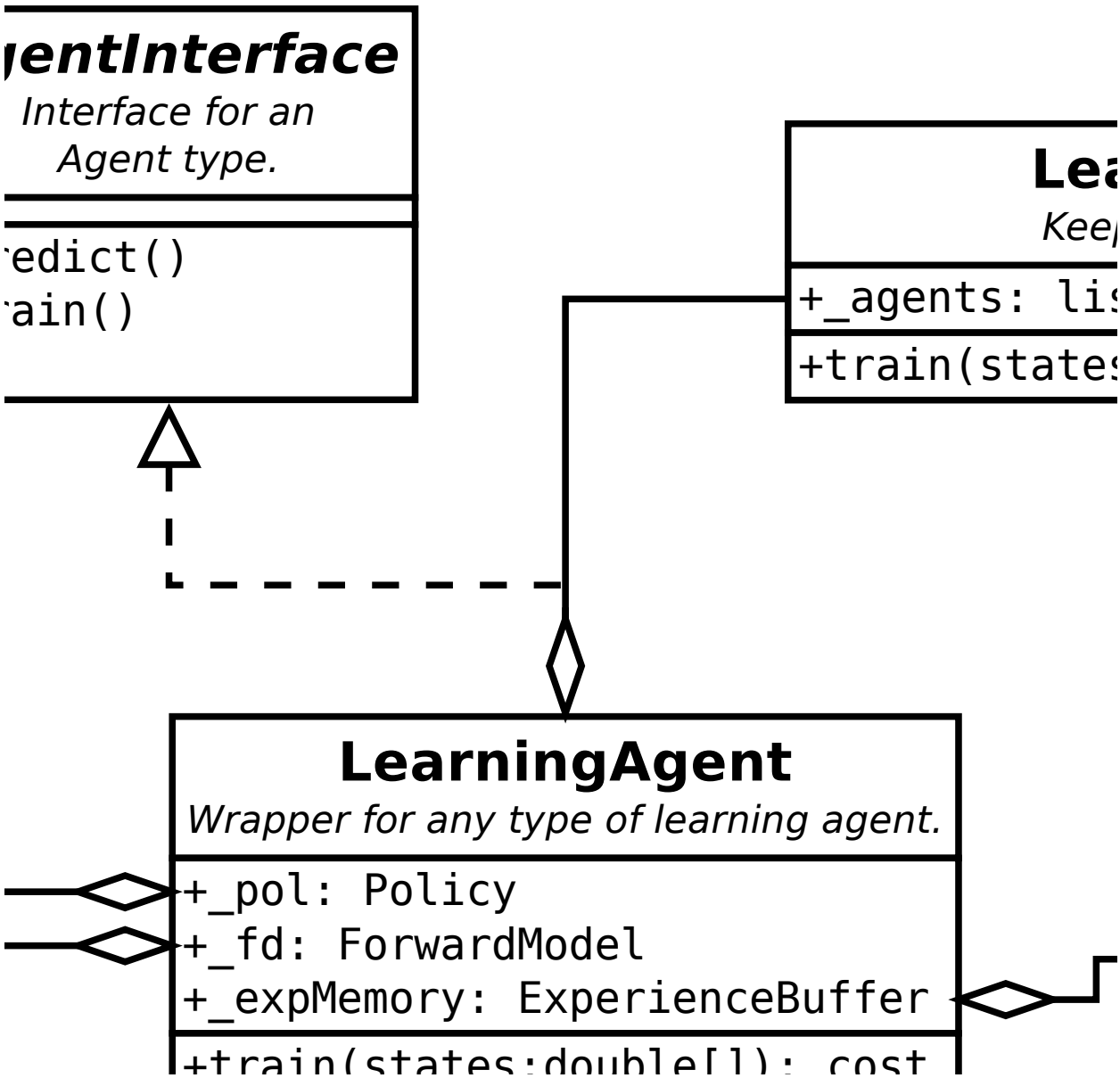
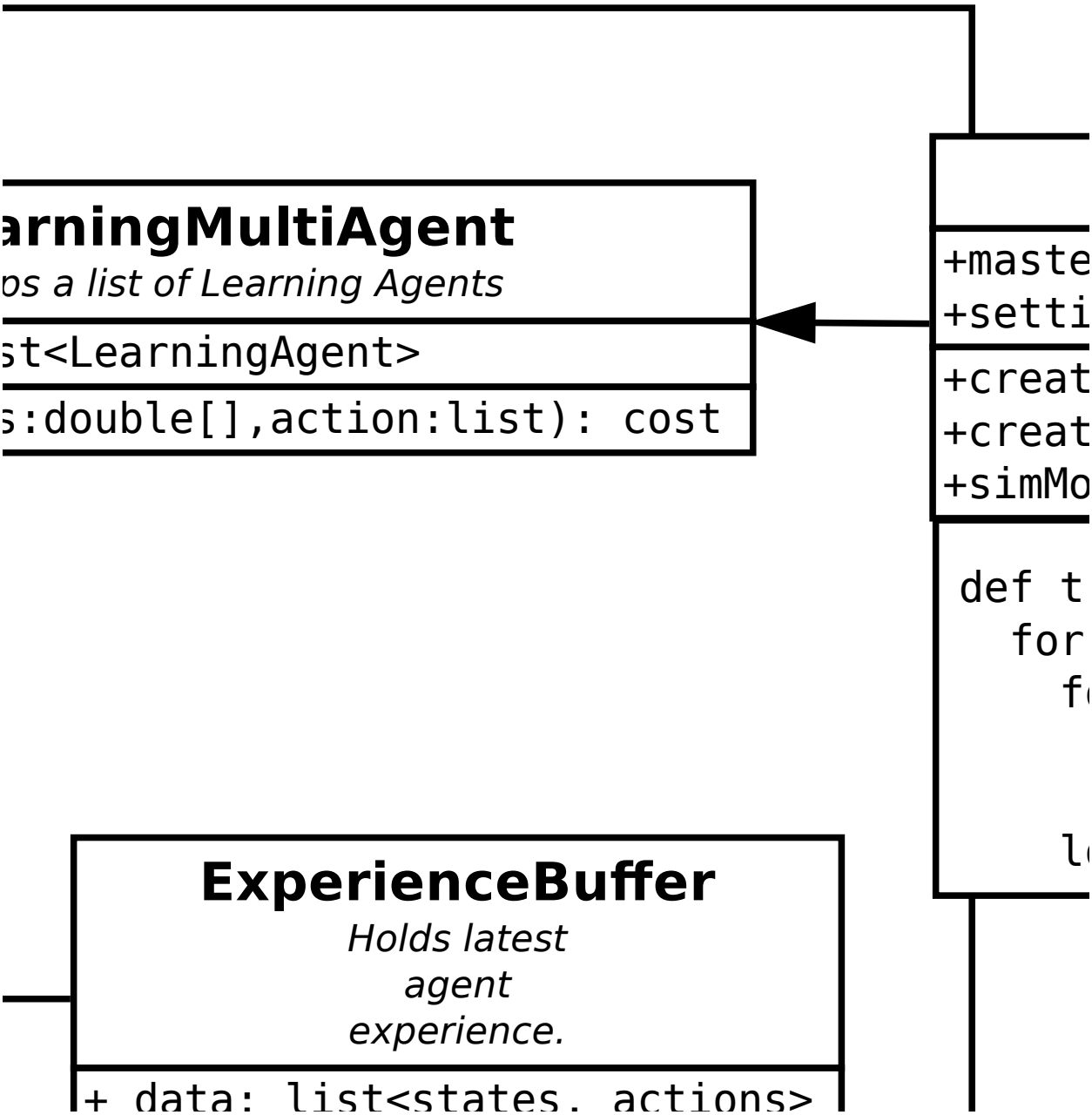
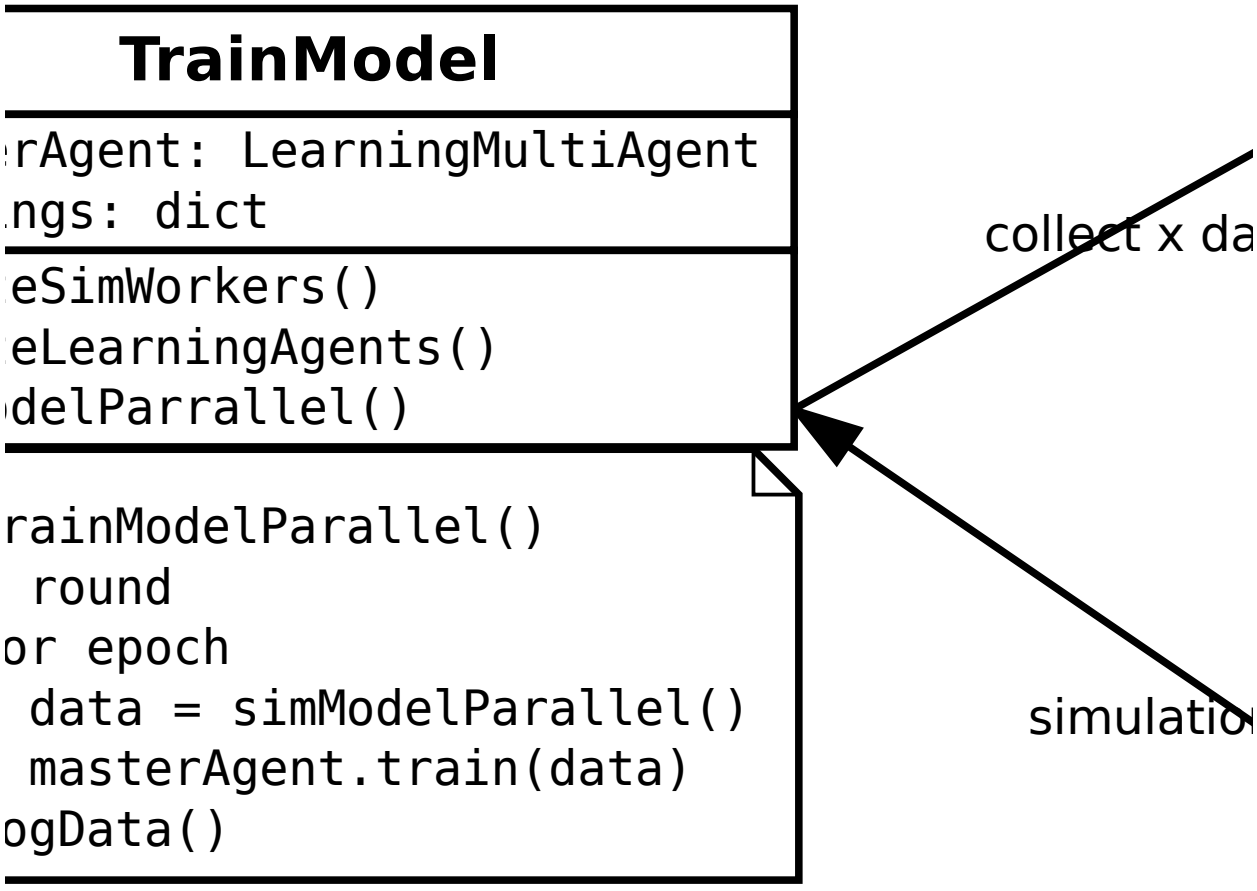
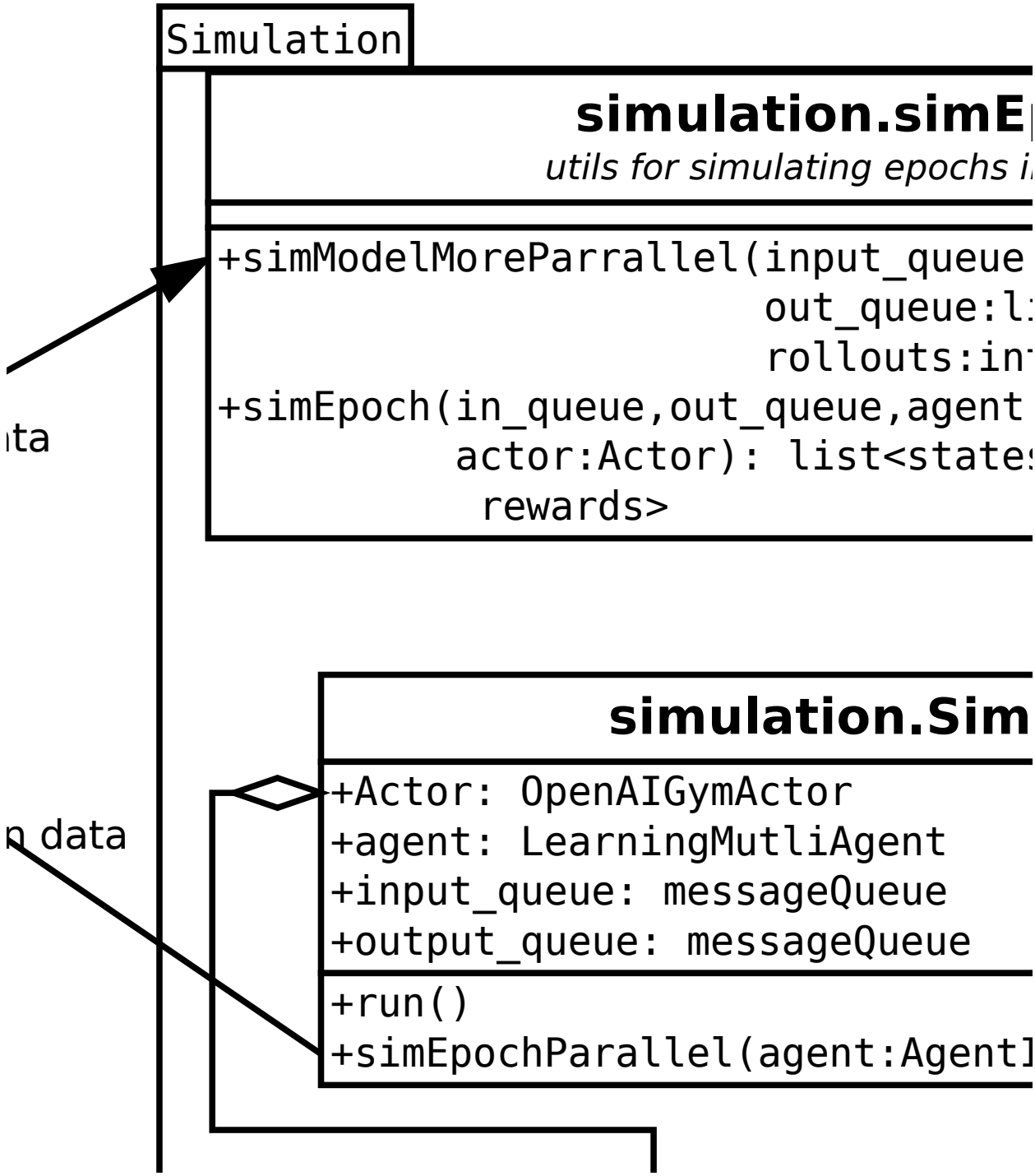
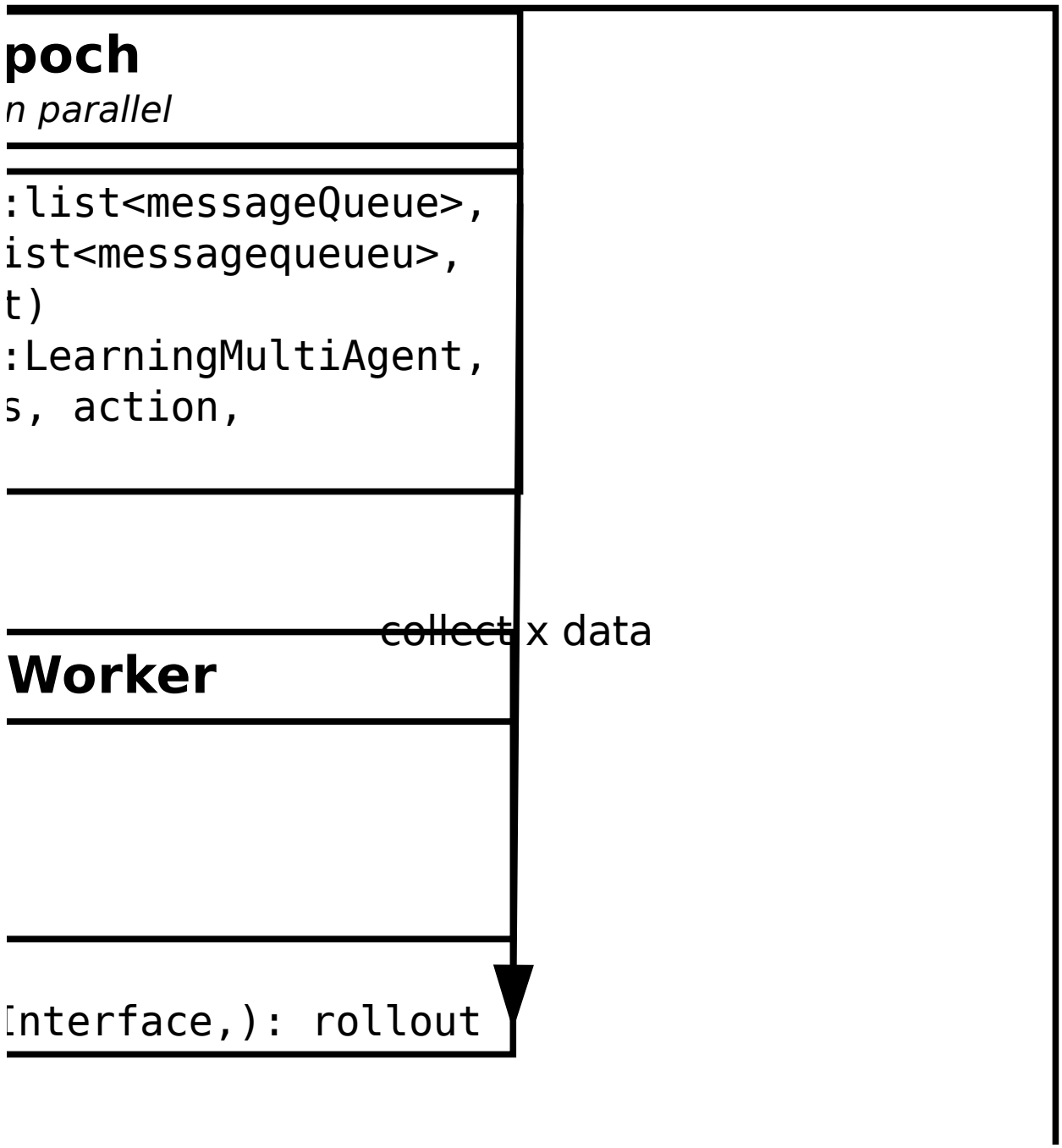

learn
Ag
+pr
+tr
+()

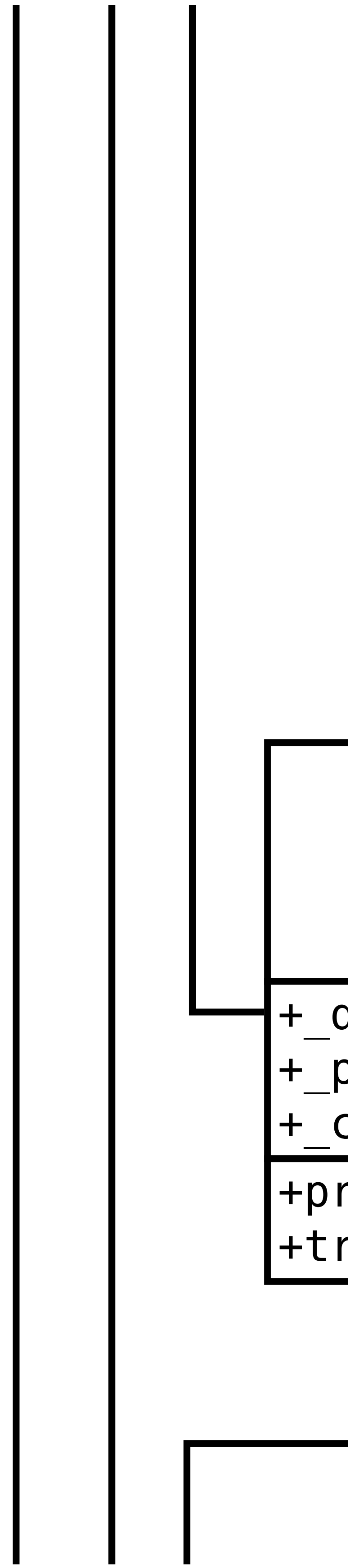

gentInterface
Interface for an
Agent type.
redict()
rain()
Lea
Kee
+_agents: lis
+train(states
LearningAgent
Wrapper for any type of learning agent.
+_pol: Policy
+_fd: ForwardModel
+_expMemory: ExperienceBuffer
+train(states:double[]): cost

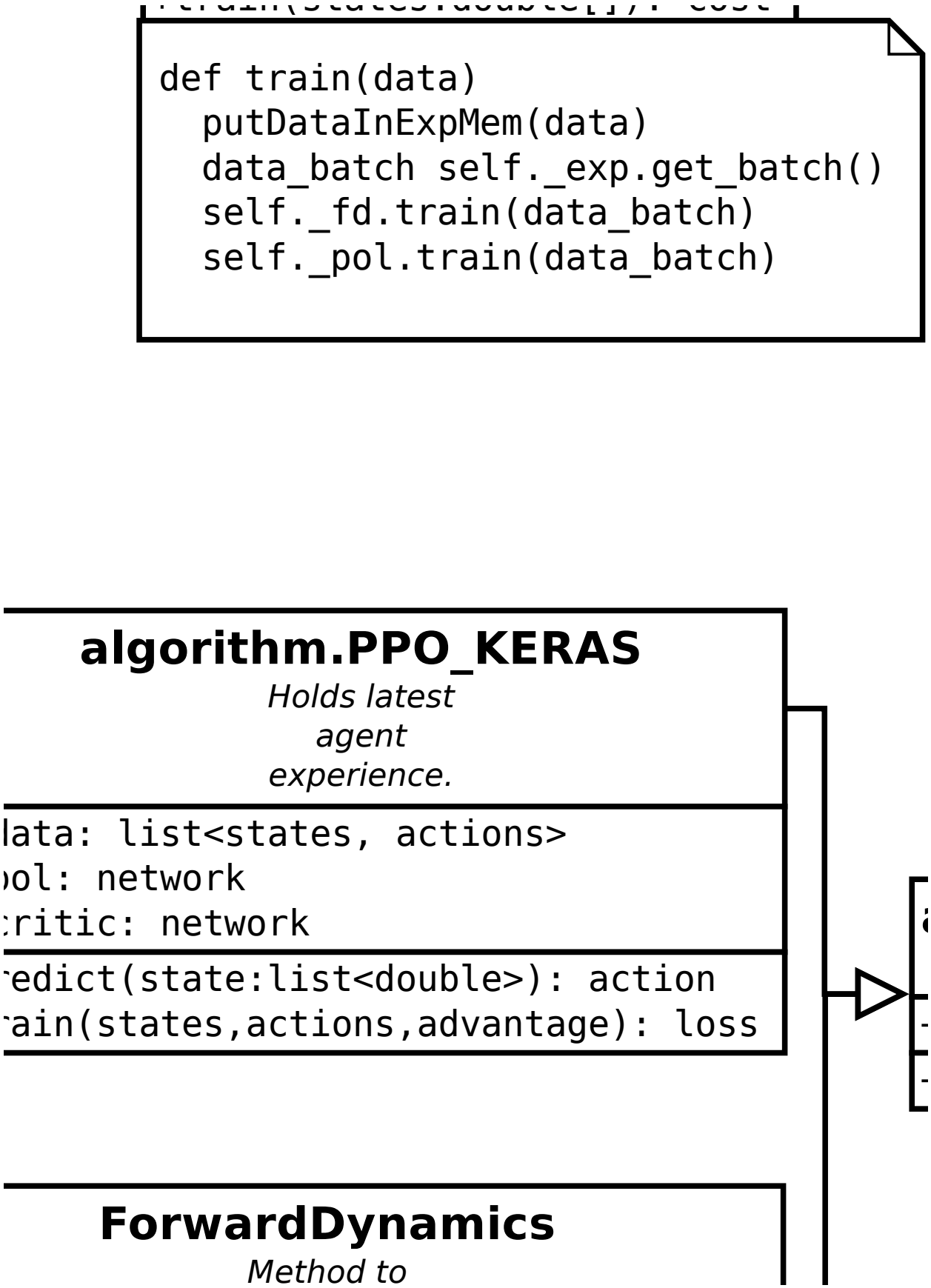

arningMultiAgent
os a list of Learning Agents
st<LearningAgent>
s:double[],action:list): cost
+maste
+setti
+creat
+creat
+simMo
def t
for
f
l
ExperienceBuffer
Holds latest
agent
experience.
+ data: list<states, actions>


TrainModel
rAgent: LearningMultiAgent
ngs: dict
eSimWorkers()
eLearningAgents()
delParrallel()
rainModelParallel()
round
or epoch
data = simModelParallel()
masterAgent.train(data)
ogData()
collect x da
simulatio


Simulation
simulation.simE
utils for simulating epochs i
+simModelMoreParrallel(input_queue
out_queue:l
rollouts:in
+simEpoch(in_queue,out_queue,agent
actor:Actor): list<state
rewards>
ta
simulation.Sim
+Actor: OpenAIGymActor
+agent: LearningMutliAgent
+input_queue: messageQueue
+output_queue: messageQueue
n data
+run()
+simEpochParallel(agent:Agent]


poch
n parallel
:list<messageQueue>,
ist<messagequeueu>,
t)
:LearningMultiAgent,
s, action,
collect x data
Worker
Interface,): rollout



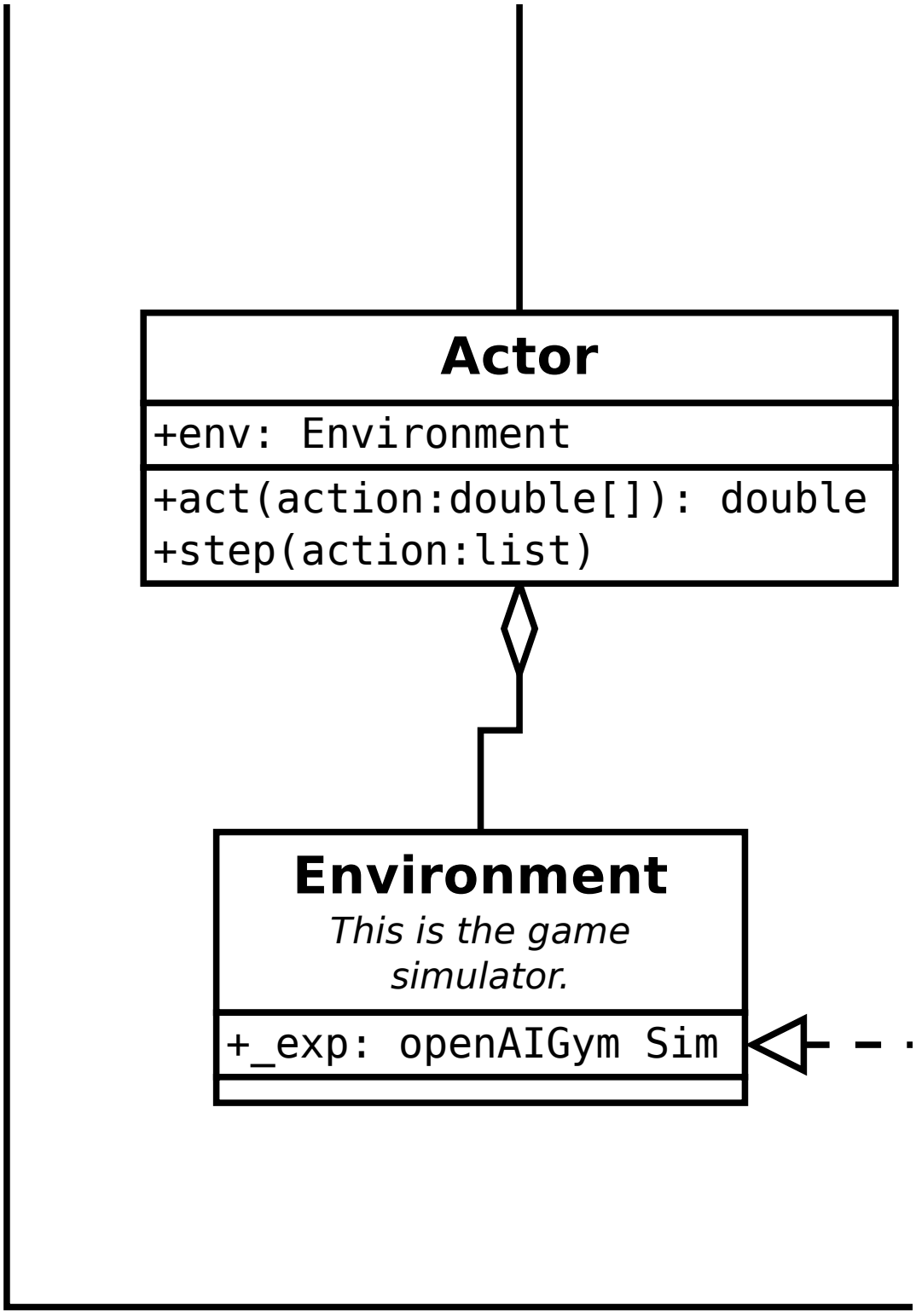

def train(data)
putDataInExpMem(data)
data_batch self._exp.get_batch()
self._fd.train(data_batch)
self._pol.train(data_batch)
algorithm.PPO_KERAS
Holds latest
agent
experience.
lata: list<states, actions>
ol: network
ritic: network
edict(state:list<double>): action
ain(states,actions,advantage): loss
ForwardDynamics
Method to

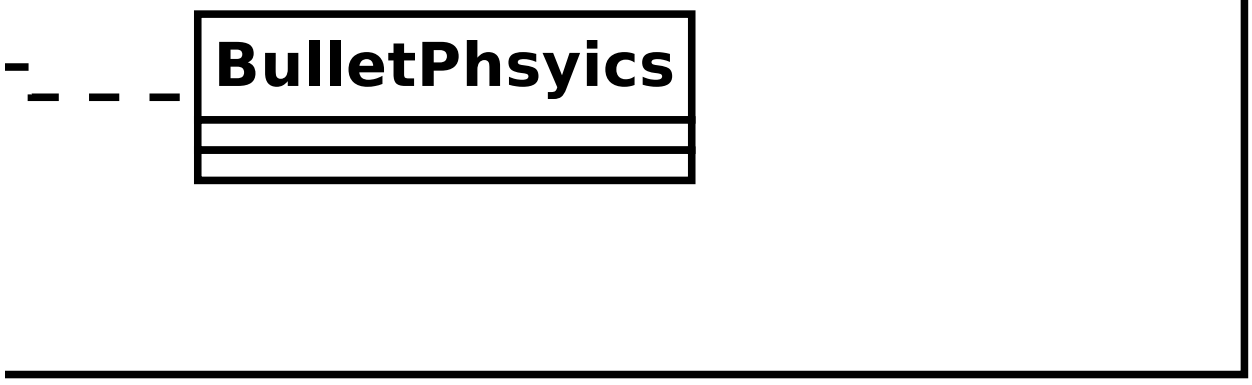
algorithm.AlgorithmInterface

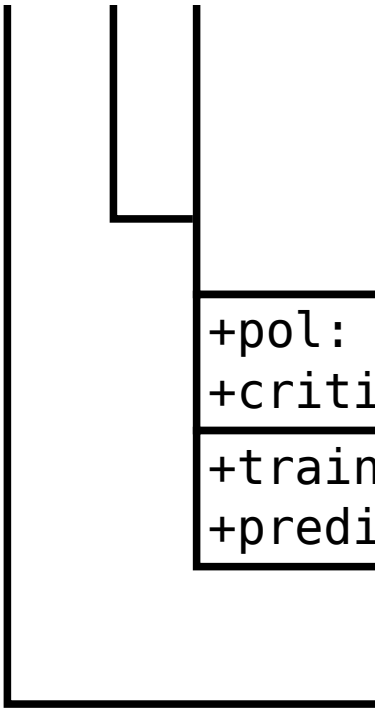
Interface for a learning algroithm

+_data: list<states, actions>

+predict(state): action


Actor
+env: Environment
+act(action:double[]): double
+step(action:list)
Environment
This is the game simulator.
+_exp: openAIGym Sim


BulletPhsyics


+pol:
+criti
+train
+predi

calculated the forward dynamics from a current state and selected action.

network
.c: network

(states,actions,next_states,reward)
.ct(states,actions)