```python
"""learner module of "AI training python"

The simulation engine to run the loaded environment.

Returns:
    None
Exports:
    "dqn_weights.h5f.index" exports an index file for the calculated weights
"""
# My standard linting settings
# pylint: disable=trailing-whitespace
# pylint: disable=logging-fstring-interpolation
# pylint: disable=line-too-long

import random
import os
# Import the Sequential model from keras
# as well as the flatten node to flatten out the 2-dimensional inputs and
# dense nodes as the default tensorflow deep leaning node
from keras.models import Sequential
from keras.layers import Dense, Flatten, Input # Flatten import unused for personal environment
from keras.optimizers import Adam

import pandas # for reading csv files

from gym import Env # for creating our own environment with gym
from gym.spaces import Discrete, Box

from rl.agents import DQNAgent # Multiple agents possible (see https://keras-rl.readthedocs.io/en/latest/) [DQN, NAF, DDPG, SARSA, CEM]
from rl.policy import BoltzmannGumbelQPolicy # Value or policy based reinforcement learning -> here: policy (BoltzmannGumbelQPolicy)
from rl.memory import SequentialMemory # To maintain memory

import numpy as np

#Custom imports:
import create_environments

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # fixing the "Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found" error, but also disables the option to run from the GPU

# Possible to use one or multiple lines from the "training_data.csv"
TRAINING_DATA_FILE = create_environments.FILENAME
LINE_FROM_ENVIRONMENT_FILE = 1

# User specific configuration
TESTING = True # should the neural network be tested after it is finished with training?
SHOW_NET_STRUCT = True # prints out the net structure on initialization of the learning process
SAVE_WEIGHTS = True # should the weights be saved after the training is finished?
SAVE_WEIGHTS_AS = "dqn_weights.h5f" # if the weights are saved, this is the filename

def point_in_rectangle(point:tuple[int, int], rect:tuple[int, int, int, int]) -> bool:
    """Global function/
    Calculate if point is in the area of a rectangle

    Args:
        point (tuple[int, int]): x, y of the point that is given
        rect (tuple[int, int, int, int]): x1, y1, x2, y2 of the rectangle to test for

    Returns:
        bool: true if point is inside, false if it is not inside
    """
    x, y = point
    x_1, y_1, x_2, y_2 = rect
    if (x_1 < x and x < x_2):
        if (y_1 < y and y < y_2):
            return True
```

```
66          return False
67
68      def point_in_circle(point:tuple[int, int], circle:tuple[int, int, int]) -> bool:
69          """Global function/
70          Calculate if point is in the area of a circle
71
72          Args:
73              point (tuple[int, int]): x, y of the point that is given
74              circle (tuple[int, int, int]): x1, y1, radius of the circle to test for
75
76          Returns:
77              bool: true if point is inside, false if it is not inside
78          """
79          x, y = point
80          x_1, y_1, rad = circle
81          if (x-x_1)**2 + (y-y_1)**2 < rad**2:
82              return True
83          return False
84
85      def calculate_aoe_hit(action_id:int, target:tuple[int, int]) -> bool:
86          """Global function/
87          Calculate if the action belonging to the action id would hit the target
88
89          Args:
90              action_id (int): action_id converted to an action with Move.moves[action_id]
91              target (tuple[int, int]): the point trying to be hit
92
93          Returns:
94              bool: true if point is hit, false if it is not hit
95          """
96          action = Move.moves[action_id]
97          square_moves = [Move.b_l, Move.b_r, Move.t_l, Move.t_r]
98          if action in square_moves:
99              return point_in_rectangle(target, Move.aoe[action])
100         else:
101             return point_in_circle(target, Move.aoe[Move.shockwave])
102
103     class Environment(Env):
104         """Gym Env Inheritance of the object of the OpenAI gym environment
105
106         Args:
107             Env (gym.Env): inheriting the gym.Env properties
108         """
109         def __init__(self, player_pos:tuple[int, int]) -> None:
110             """The constructor function for an custom Environment
111
112             Args:
113                 player_pos (tuple[int, int]): the player position for which the
                    environment is set up
114             """
115             # Inherit from Env
116             super().__init__()
117             # self.observation_space = Box(low=np.array(playfield_zeros,
                dtype=np.float32), high=np.array(playfield_max, dtype=np.float32),
                dtype=np.float32) # action array
118             self.observation_space = Box(low=0, high=1, shape=(410, 410), dtype=np.uint8)
119             self.action_space = Discrete(5, start=0) # actions we can take (Move.moves)
120             # LEGACY: using indexed state variable
121             # self.state = random.choice([Move.shockwave, Move.b_l, Move.b_r, Move.t_l,
                Move.t_r]) # set start action
122             self.state = random.randint(0, 4)
123             self.player_position = player_pos
124             self.player_position_move = 10
125
126         def step(self, action:int) -> tuple[int, int, bool, dict]:
127             """overwriting the step function from the gym.Env class
128             # would be possible to user super().__init__() for perfect implementation,
                but is simply not required here
129
130             Args:
131                 action (int): action index for the self.state variable
132
```

```
133             Returns:
134                 tuple[int, int, bool, dict]: current state, reward (either -1 or 1), done
                    (true when player movement was simulated 10 times)
135             """
136             # LEGACY: using indexed state variable
137             # self.state = Move.moves[action]
138             self.state = [[0 for x in range(410)] for y in range(410)]
139             self.state[self.player_position[0]][self.player_position[1]] = 1
140             self.player_position_move -= 1
141             # print(self.state, self.player_position) # DEBUGGING
142             # Add player move noise
143             self.player_position = self.player_position[0]+random.randint(-self.
                player_position_move, self.player_position_move), self.player_position[1]+
                random.randint(-self.player_position_move, self.player_position_move)
144             if calculate_aoe_hit(action, self.player_position):
145                 self.state = [[0 for x in range(410)] for y in range(410)]
146                 self.state[self.player_position[0]][self.player_position[1]] = 1
147                 reward = 1
148             else:
149                 reward = -1
150             if self.player_position_move <= 0:
151                 done = True
152             else:
153                 done = False
154 
155             info = {} # placeholder (required by OpenAI)
156 
157             return self.state, reward, done, info
158 
159         def render(self):
160             """placeholder for a possible implementation with pygame or tkinter. Not done
                for time reasons and the non necessity
161             """
162 
163         def reset(self) -> int:
164             # Number of parameters was 3 in 'Env.reset' and is now 1 in overridden
                'Environment.reset' method
165             """called periodically after each iteration of the specified step amount
166 
167             Returns:
168                 int: current state to pass onto the next interval
169             """
170             # LEGACY: using indexed state variable
171             # self.state = random.choice([Move.shockwave, Move.b_l, Move.b_r, Move.t_l,
                Move.t_r]) # set start action (same as above)
172             self.state = random.randint(0, 4)
173             self.player_position_move = 10 # Reset player move noise
174             return self.state
175 
176     #Dataclass
177     class Move():
178         """dataclass for specifying the possible moves and packing them into lists and
            dictionaries
179         @dataclass decorator not used since it would be overkill
180         """
181         shockwave = "shockwave"
182         b_l = "bottom_left"
183         b_r = "bottom_right"
184         t_l = "top_left"
185         t_r = "top_right"
186 
187         moves = [shockwave, b_l, b_r, t_l, t_r]
188 
189         aoe = {"bottom_left":(0, 0, 205, 205),        # define the area of effect to
            calculate if a non-perfect move would still succeed
190                "bottom_right":(205, 0, 410, 205),      # point defined like (x_1, y_1,
                   x_2, y_2)
191                "top_left":(0, 205, 205, 410),
192                "top_right":(205, 205, 410, 410),
193                "shockwave":(205, 205, create_environments.RADIUS)}    # circle defined
                   like (middle_x, middle_y, radius)
194 
```

```python
class Agent():
    """agent class used to build a custom agent object
    """
    def __init__(self) -> None:
        """self.env creates our custom environment for the specified x, y start position of the player.
        Also possible to iterate over them using a for loop, for utilizing the generated training_data in the training_data.csv
        """
        self.training_data = pandas.read_csv(TRAINING_DATA_FILE)

        self.env = Environment((254, 82))
        # print(self.env.step(4)) # DEBUGGING


    def build_model(self) -> Sequential: # actions:int
        """function for building a keras model

        Args:
            actions (int): amount of actions to build the model for (in this case 5)

        Returns:
            Sequential: returns the Sequential class of the model created by the keras module
        """
        states = self.env.observation_space.shape
        actions = self.env.action_space.n
        # print(states, actions) # DEBUGGING
        model = Sequential()
        # LEGACY: no need to flatten out the custom env with an input_shape before usage # TODO
        # model.add(Flatten(input_shape=states))
        model.add(Input(shape=(410, 410, 5)))
        model.add(Dense(256, activation="relu")) # Dense node layer as standard keras neuron to generate deep reinforcement learning algorithms
        model.add(Dense(128, activation="relu"))
        model.add(Dense(64, activation="relu"))
        model.add(Dense(32, activation="relu"))
        model.add(Dense(8, activation="relu"))
        model.add(Flatten())
        model.add(Dense(actions, activation="softmax"))
        model.summary()
        print(model.input_shape)
        return model

    def build_agent(self, model:Sequential) -> DQNAgent:
        """function for building a keras model

        Args:
            model (Sequential): _description_

        Returns:
            DQNAgent: personal DQNAgent (see line 20 for more info "from rl.agents import DQNAgent")
        """
        actions = self.env.action_space.n # = 5
        policy = BoltzmannGumbelQPolicy()
        memory = SequentialMemory(limit=50_000, window_length=1)
        print(model.output_shape)
        dqn = DQNAgent(model=model, memory=memory, policy=policy, nb_actions=actions, nb_steps_warmup=20_000, target_model_update=1e-2)
        return dqn


if __name__ == "__main__":
    # All personal objects are labeled with the prefix "my_"
    my_agent = Agent()
    # LEGACY: env is created in the __init__ of Agent
    # env = agent.create_environment("(254, 82)","shockwave")
    my_model = my_agent.build_model()
    # my_model.summary()
    print("here")
```

```
260         my_dqn = my_agent.build_agent(my_model)
261
262         my_dqn.compile(Adam(learning_rate=0.01), metrics=["mae"]) # mae = mean absolute
            error
263         my_dqn.fit(my_agent.env, nb_steps=50_000)
264         if TESTING:
265             scores = my_dqn.test(my_agent.env, nb_episodes=100, visualize=False)
                #nb_episodes = amount of testing episodes to run
266             print(np.mean(scores.history["episode_reward"]))
267         if SAVE_WEIGHTS:
268             my_dqn.save_weights(SAVE_WEIGHTS_AS, overwrite=True)
269
```