Mars Lander - Open ai enviroment

Generated by Doxygen 1.8.13

Contents

Chapter 1

Mars Lander - Open ai enviroment

Made By Michi, Zied and Zhen

About

mars landing an Open ai test environment. About this environment: The goal is to land at the base station with the satelite, on the way to the station the satelite must avoid obstacles like black holes or asteroids.

The satelite starts from an random position somewhere in the upper lefter corner and flies with an constant speed in direction of the station.

It contains the standart Open AI enviroment methods :

- reset()
- step() We hope you guys know how to use these

  The enviroment is solved if you hit the space station five times in a row. A flag will be set if this is the case (there will also be a console output). The flag is part of step return (info == True) if solved.

Important for usage!

For setting the level of dificulty init the nviroment like this :

```
env = gym.make('mars_lander-v0', level=1)
```

where level is 1, 2 or 3

ALSO you have to render the env youself add something like this to your agent if you want

```
if render:
    still_open = self.env.render()
    if still_open is False:
        break
```

Observations

This environment consists of several levels. At level 1 there are no obstacles, at level 2 there are obstacles and at level 3 there are obstacles that always appear and move around. for level 1 is the observation space:

- s[0] x position of the satelite
- s[1] y position of the saddle rivet
- s[2] x position of the base station
- s[3] y position of the base station

for level 2 and three the positions of the obstacles are transferred to the observations of 1

- s[4] x position of the asteroid
- s[5] y position of the asteroid
- s[6] x position of the black hole_1
- s[7] y position of the black hole_1
- s[8] x position of the black hole_2
- s[9] y position of the black hole_2

Actions

The possible actions of the satelite are discrete, either it activates the motors (action = 1) or it deactivates them (action = 0).

Reward

The reward is sparse: (1) For hitting the space station the agent gets +50 (2) For hitting an obstacle he gets -50 (3) For crushing in the ground he gets 25 - distance to the lander

Virtual enviroment

How to use :

- First you have to init a venv on your desktop we do not want to push it on git for this run:

```
python3 -m venv venv
```

- After that activate venv by running and install pip-tools

```
source venv/bin/activate
python3 -m pip install pip-tools
```

- Then compile requirements.in with command line :

```
pip-compile dependencies/requirements.in
```

- pip-compile requirements.in –> CREATES requirements.txt
- this file can be used in venv to install all requirements

```
pip install -r dependencies/requirements.txt
```

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Namespace Documentation

5.1 group Namespace Reference

-01::learning::agent_d

5.1.1 Detailed Description

-01::learning::agent_d

-01::game::mars_lander

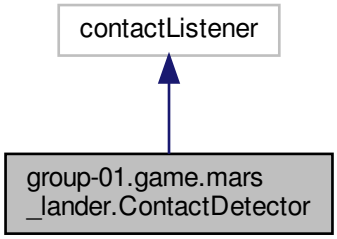
```
File contains different classes that handle reinforcement learning implementing Sarsa and Sarsa lambda


mars landing an Open ai test environment.
About this environment:
The goal is to land at the base station with the satelite, on the way to the station the satelite must avoid c
like black holes or asteroids.
The possible actions of the satelite are discrete,
either it activates the motors (action = 1) or it deactivates them (action = 0).
This environment consists of several levels. At level 1 there are no obstacles,
at level 2 there are obstacles and at level 3 there are obstacles that always appear and move around.
for level 1 is the observation space:
s[0] x position of the satelite
s[1] y position of the saddle rivet
s[2] x position of the base station
s[3] y position of the base station

for level 2 and three the positions of the obstacles are transferred to the observations of 1
s[4] x position of the asteroid
s[5] y position of the asteroid
s[6] x position of the black hole_1
s[7] y position of the black hole_1
s[8] x position of the black hole_2
s[9] y position of the black hole_2

The reward is sparse:
(1) For hitting the space station the agent gets +50
(2) For hitting an obstacle he gets -50
(3) For crushing in the ground he gets 25 - distance to the lander
```

Chapter 6

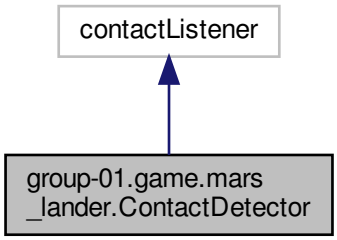
Class Documentation

6.1 group-01.game.mars_lander.ContactDetector Class Reference

Inheritance diagram for group-01.game.mars_lander.ContactDetector:



Collaboration diagram for group-01.game.mars_lander.ContactDetector:



Public Member Functions

- def **__init__** (self, env)
- def BeginContact (self, contact)
- def **EndContact** (self, contact)

Public Attributes

- **env**

6.1.1 Detailed Description

```
Class for detecting and handle contacts between the different bodys.
To handle the different contact classes we split the objects in 4 groups:
(1) Satelite
(2) Surface
(3) Asteroid + blackhole
(4) Station
For the different contacts we get different rewards. We are mainly interested in contacts
between 1 and the other objects.
```

6.1.2 Member Function Documentation

6.1.2.1 BeginContact()

```
def group-01.game.mars_lander.ContactDetector.BeginContact (
              self,
              contact )
```

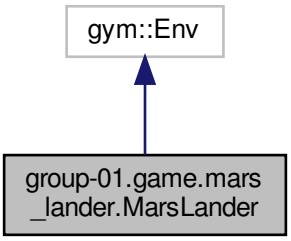
```
Detects and handels collisions between the different objects
```

The documentation for this class was generated from the following file:

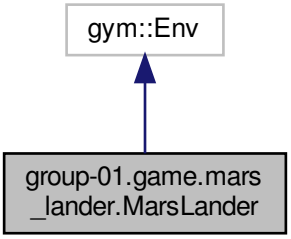
- game/mars_lander.py

6.2 group-01.game.mars_lander.MarsLander Class Reference

Inheritance diagram for group-01.game.mars_lander.MarsLander:



Collaboration diagram for group-01.game.mars_lander.MarsLander:



Public Member Functions

- def **__init__** (self, level=1)
- def set_level (self, level=1)
- def reset (self)
- def step (self, p_action)
- def render (self, mode='human')
- def **close** (self)

Public Attributes

- **viewer**
- **satelit_pic_path**
- **station_pic_path**
- **asteroid_pic_path**

- **blackhole_pic_path**
- **rotation_counter**
- **mars**
- **surface**
- **lander**
- **last_rewards**
- **reset_counter**
- **solved_problem**
- **OBSERVATIONS**
- **observation_space**
- **action_space**
- **game_over**
- **collision**
- **prev_shaping**
- **done**
- **reward**
- **helipad_x1**
- **helipad_x2**
- **helipad_y**
- **sky_polys**
- **mars_polys**
- **asteroid**
- **pos_a**
- **pos_b**
- **blackhole**
- **blackhole_2**
- **station**
- **legs**
- **drawlist**
- **black_1_pic**
- **black_2_pic**
- **ast_img**
- **sat_img**
- **station_img**

Static Public Attributes

- dictionary **metadata**

6.2.1 Detailed Description

```
This is the main class handeling the game and the enviroment.
```

6.2.2 Member Function Documentation

6.2.2.1 render()

```
def group-01.game.mars_lander.MarsLander.render (
              self,
              mode = 'human' )
```

Rendering method. displays the enviroment so you can see what is happening.

6.2.2.2 reset()

```
def group-01.game.mars_lander.MarsLander.reset (
              self )
```

Method that resets the enviroment should be executed before every new episode.
@return : state, reward, done, info (True when hitting the space station 10 times in a row)

6.2.2.3 set_level()

```
def group-01.game.mars_lander.MarsLander.set_level (
              self,
              level = 1 )
```

This method sets the values for the different difficulty levels

6.2.2.4 step()

```
def group-01.game.mars_lander.MarsLander.step (
              self,
              p_action )
```

Step method executes a single step in the enviroment.
@param p_action : action that should be executed 0 or 1.
@return : state, reward, done, info (True when hitting the space station 10 times in a row)

6.2.3 Member Data Documentation

6.2.3.1 metadata

dictionary group-01.game.mars_lander.MarsLander.metadata [static]

Initial value:

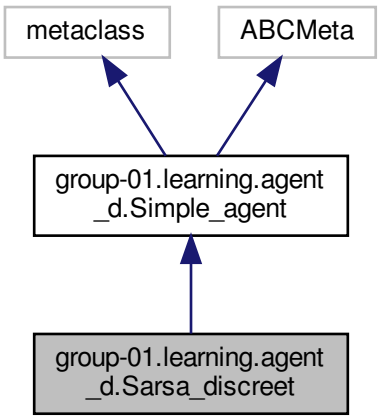
```
= {
        'render.modes': ['human', 'rgb_array'],
        'video.frames_per_second': FPS
    }
```

The documentation for this class was generated from the following file:

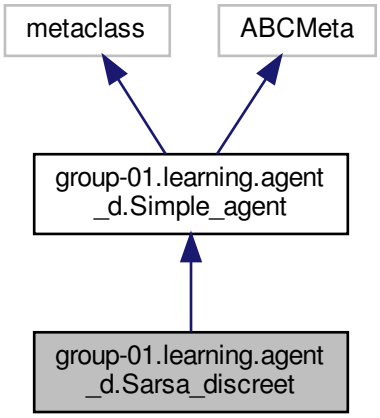
- game/mars_lander.py

6.3 group-01.learning.agent_d.Sarsa_discreet Class Reference

Inheritance diagram for group-01.learning.agent_d.Sarsa_discreet:



Collaboration diagram for group-01.learning.agent_d.Sarsa_discreet:



Public Member Functions

- def **__init__** (self, env, logging_level=logging.INFO)
- def learn (self, episodes, epsilon=0.1, epsilon_decay=0.99, epsilon_min=0.05, gamma=0.5, alpha=0.1, render=True)
- def shape_reward (self, reward)
- def update (self, state, action, reward, next_state, next_action=None)

Public Attributes

- **epsilon**
- **epsilon_decay**
- **epsilon_min**
- **alpha**
- **gamma**
- **rewards**
- **solved**

6.3.1 Detailed Description

```
Sarsa Lambda reinforcement learning algorithem (according to Barto and Suttons Reinforcement learning an intor
```

6.3.2 Member Function Documentation

6.3.2.1 learn()

```
def group-01.learning.agent_d.Sarsa_discreet.learn (
              self,
              episodes,
              epsilon = 0.1,
              epsilon_decay = 0.99,
              epsilon_min = 0.05,
              gamma = 0.5,
              alpha = 0.1,
              render = True )
```

```
SARSA : reinforcement learning algorithem
@param episodes : number of episodes we want to train
@param epsilon : epsilon value for epsilon greedy policy
@param epsilon_decay : decay value of epsilon for each episode
@param epsilon_min : min epsilon value
@param gamma : gamma value regulates the influence of the following state
@param alpha : learning rate
@param render : boolean value if we want to render the enviroment
```

6.3.2.2 shape_reward()

```
def group-01.learning.agent_d.Sarsa_discreet.shape_reward (
              self,
              reward )
```

```
Method that handels reward shaping
```

6.3.2.3 update()

```
def group-01.learning.agent_d.Sarsa_discreet.update (
              self,
              state,
              action,
              reward,
              next_state,
              next_action = None )
```

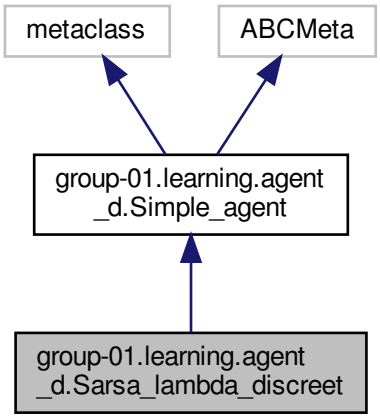
```
Update Q table according to following formula:
Q(s,a) = Q(s,a) + alpha [reward + gamma * Q(next_s, next_a) - Q(s,a)]
@param state : Actual state of the agent
@param action : action the agent takes at state 'state'
@param reward : reward the agent gets for taking action 'action' in state 'state'
@param next_state : state of the agent after taking action
@param next_action : action to take in state 'next_state'
```

The documentation for this class was generated from the following file:

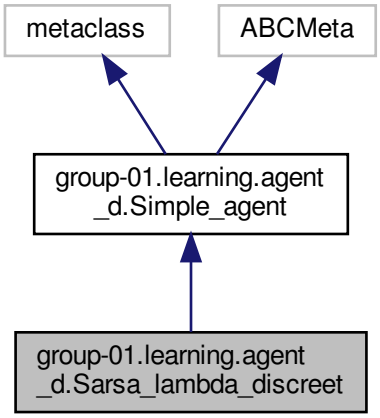
- learning/agent_d.py

6.4 group-01.learning.agent_d.Sarsa_lambda_discreet Class Reference

Inheritance diagram for group-01.learning.agent_d.Sarsa_lambda_discreet:



Collaboration diagram for group-01.learning.agent_d.Sarsa_lambda_discreet:



Public Member Functions

- def **__init__** (self, env, logging_level=logging.INFO)
- def learn (self, episodes, epsilon=0.1, epsilon_decay=0.99, epsilon_min=0.05, gamma=0.5, alpha=0.↩ 1, lambd=0.5, render=True)
- def **shape_reward** (self, reward)
- def **update** (self, state, action, reward, next_state, next_action=None)

Public Attributes

- **epsilon**
- **epsilon_decay**
- **epsilon_min**
- **alpha**
- **gamma**
- **lambd**
- **rewards**
- **solved**

6.4.1 Detailed Description

```
Sarsa Lambda reinforcement learning algorithem (according to Barto and Suttons Reinforcement learning an intor
```

6.4.2 Member Function Documentation

6.4.2.1 learn()

```
def group-01.learning.agent_d.Sarsa_lambda_discreet.learn (
              self,
              episodes,
              epsilon = 0.1,
              epsilon_decay = 0.99,
              epsilon_min = 0.05,
              gamma = 0.5,
              alpha = 0.1,
              lambd = 0.5,
              render = True )
```

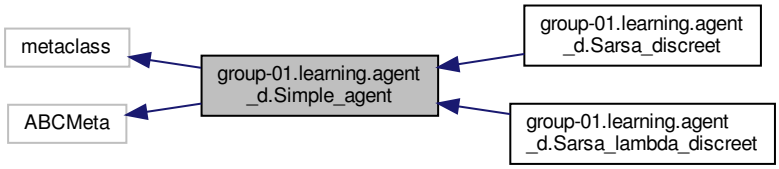
```
SARSA lambda: reinforcement learning algorithem
@param episodes : number of episodes we want to train
@param epsilon : epsilon value for epsilon greedy policy
@param epsilon_decay : decay value of epsilon for each episode
@param epsilon_min : min epsilon value
@param gamma : gamma value regulates the influence of the following state
@param alpha : learning rate
@param render : boolean value if we want to render the enviroment
@param lambd : lambda value defines influence of visited states
```

The documentation for this class was generated from the following file:

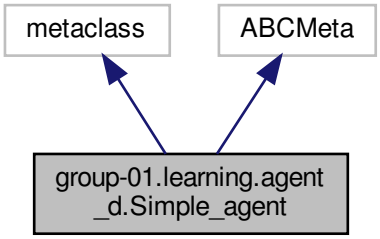
- learning/agent_d.py

6.5 group-01.learning.agent_d.Simple_agent Class Reference

Inheritance diagram for group-01.learning.agent_d.Simple_agent:



Collaboration diagram for group-01.learning.agent_d.Simple_agent:



Public Member Functions

- def **__init__** (self, env, logfile_name, logging_level)
- def get_epsilon_greedy_policy (self, epsilon)
- def after_train (self, reward_list)
- def learn (self, episodes, epsilon=0.1, epsilon_decay=0.99, epsilon_min=0.05, lambd=0.0, gamma=0.↩ 5, alpha=0.1, render=True)
- def update (self, d_state, action, reward, d_next_state, next_action)

Public Attributes

- **env**
- **estimator**

6.5.1 Detailed Description

```
Abstract class defining basic methods and attributes for learning.
```

6.5.2 Member Function Documentation

6.5.2.1 after_train()

```
def group-01.learning.agent_d.Simple_agent.after_train (
              self,
              reward_list )
```

Method executing standart plots after training

6.5.2.2 get_epsilon_greedy_policy()

```
def group-01.learning.agent_d.Simple_agent.get_epsilon_greedy_policy (
              self,
              epsilon )
```

Method returning an epsilon greedy policy

6.5.2.3 learn()

```
def group-01.learning.agent_d.Simple_agent.learn (
              self,
              episodes,
              epsilon = 0.1,
              epsilon_decay = 0.99,
              epsilon_min = 0.05,
              lambd = 0.0,
              gamma = 0.5,
              alpha = 0.1,
              render = True )
```

Abstract method define the learning of the agent

6.5.2.4 update()

```
def group-01.learning.agent_d.Simple_agent.update (
              self,
              d_state,
              action,
              reward,
              d_next_state,
              next_action )
```

```
Abstract method define the Q update
```

The documentation for this class was generated from the following file:

- learning/agent_d.py

6.6 group-01.learning.agent_d.State_estimator Class Reference

Public Member Functions

- def **__init__** (self)
- def get_eglibility_trace (self, state, action)
- def update_eglibility_trace (self, state, action, value)
- def get_q_value (self, state, action=None)
- def update_q_value (self, state, action, value)

Public Attributes

- **q_table**
- **eg_trace**

6.6.1 Detailed Description

```
Class to encode the continous features to discrete values, we achive this by putting a grid on the enviroment.
```

6.6.2 Member Function Documentation

6.6.2.1 get_eglibility_trace()

```
def group-01.learning.agent_d.State_estimator.get_eglibility_trace (
              self,
              state,
              action )
```

```
Returns the value of the eligibility trace for state, action pair
```

6.6.2.2 get_q_value()

```
def group-01.learning.agent_d.State_estimator.get_q_value (
              self,
              state,
              action = None )
```

Returns the Q value for state, action pair

6.6.2.3 update_eglibility_trace()

```
def group-01.learning.agent_d.State_estimator.update_eglibility_trace (
              self,
              state,
              action,
              value )
```

Sets the value of the eligibility trace at position state, action to value

6.6.2.4 update_q_value()

```
def group-01.learning.agent_d.State_estimator.update_q_value (
              self,
              state,
              action,
              value )
```

Updates Q tabel entry at position state, action to value

The documentation for this class was generated from the following file:

- learning/agent_d.py

Index