FactoryRally-Receiver

version 1.0

Julian Kolesik

April 04, 2021

Contents

Welcome to FactoryRally-Sender’s documentation!

Hardware_Main module

class Hardware_Main.**HardwareMain**
Bases: **object**
This class initiates the REST Receiver and the MQTT Sender and starts the game process.

setup_connection_handler ()
This method initializes the connection handler.

setup_resource_handler ()
This method initializes the resource handler.

Hardware_Main.**reset** ()
This method performs a reset upon game end.

GUI package

Submodules

GUI.GameGUI module

class GUI.GameGUI.**GameGUI**
Bases: **tkinter.Tk**
This class provides a simple interface for the user to choose a game.

show_frame (cont**)**
This method is used to switch between frames.

Parameters: **cont** – the name of the frame which should be switched to

class GUI.GameGUI.**GameSelector** (parent, controller)
Bases: **tkinter.Frame**
This class provides a simple interface for the user to choose a game.

return_game ()
This method retrieves the selected game when the user presses the button.

set_games (games**)**
This method sets a given list.

Parameters: **games** – the currently active games

class GUI.GameGUI.**GameStartPage** (parent, controller)
Bases: **tkinter.Frame**
This class is used to tell it should generate a new Publisher as a new game started.

ACTIVE = *False*

button_click ()
This method sets the parameter active to true if a game is running according to user input.

get_state ()
This method returns the current state of the button. If it is pressed, it resets it.

Returns: whether or not the button is pressed

class GUI.GameGUI.**InformationDisplay** (parent, controller)
Bases: **tkinter.Frame**
This class is used to display the current game event on the Raspberry Pi.

update_information (msg)
This method displays the current game event.

Param: msg: the current game event

GUI.NetworkUI module

Module contents

MQTT package

Submodules

Module contents

MQTT.MQTTPublisher module

class MQTT.MQTTPublisher.**MQTTPublisher** (gui, connection_handler, resource_handler)
Bases: **object**
This class is the MQTT Sender which pushes the current game event to the broker with the according topic.

ACTIVE = *False*

GAME_STOP = *False*

GEN_TOPIC = *'general'*

SETUP = *False*

broker = *'broker.emqx.io'*

client_id = *'python-mqtt-5949'*

close_game ()
This method unsubscribes from all current clients/topics when a game ended.

connect_mqtt ()
This method creates a connection to the MQTT Broker.

Returns: a client instance

discover_and_notify ()
This method handles the discovery of all clients in the current game and performs a mapping so each robots has its virtual id.

generate_game ()
This function generates REST Receivers for the given game. :return: a REST Receiver Instance

ids = *[]*

perform_game_start ()
This method performs a fresh start for a user selected game. It it also called when a game is finished and a new game was chosen.

port = *1883*

publish ()
This method publishes the current message to the broker with the according topic if a game is currently active.

start ()
This method starts the discover process, after that starts the client loop and publishing.

topics = *[]*

MQTT.MQTTPublisher.**evaluate_relevance** (msg)
This function evaluates whether the message gets transported or not aka if a real robot can perform it.

Parameters: **msg** – the received message to check
Returns: relevant or not

NetworkUtilities package

Submodules

NetworkUtilities.NetworkUtility module

NetworkUtilities.NetworkUtility.**connect_to_wlan** (ssid, password)
This functions connects to a network.

Parameters:
- **ssid** – the SSID of the network
- **password** – the WPA passphrase of the given network

Returns: result of the command, e.g. if connection was successful

NetworkUtilities.NetworkUtility.**evaluate_result** (result)
This function evaluates whether the connection to a network was a success or failure.

Parameters: **result** – the output of the
Returns: a list of success and msg

NetworkUtilities.NetworkUtility.**return_all_wifi_connections** ()
This function returns all currently available wifi access points.

Returns: list with all wifi SSIDs

Module contents

REST package

Submodules

REST.ConnectionHandler module

class REST.ConnectionHandler.**ConnectionHandler** (api)
Bases: **object**
This class handles the connection to the API and manages different possible errors as well as it performs checks.

api_root_url = *'http://localhost:5050/'*

game_started = *'PLAYING'*

wait_for_api_availability ()
This function waits until the API is reachable.

wait_for_initialized_game ()
This function waits until at least one game is initialized.

wait_for_running_game (game_id, resource_handler)
This function waits until a game is started.

Parameters:
- **game_id** – the given game
- **resource_handler** – the resource handler

REST.RESTClient module

class REST.RESTClient.**RestReceiver** (res, conn, game_id)
Bases: **object**
This class provides the interface to access the REST API and process the retrieved information.

EXECUTION_PHASE = *False*

PLAYING_STATE = *'PLAYING'*

check_if_all_player_have_entity ()
This method checks if all given players have their entity assigned.

Returns: whether or not all players have an entity

evaluate_correct_topic (msg)
This method returns the corresponding topic of the given message.

Returns: the controlling player of the given entity

event_types = *['movement', 'upgrade purchase', 'fill register', 'activate upgrade', 'lazer shot', 'game start', 'clear shop', 'fill shop', 'register clear', 'programming timer start', 'programming timer stop', 'random card distribution', 'take card event', 'activate checkpoint', 'game_phase_changed', 'game_round_phase_changed', 'pause', 'unpause', 'damage', 'lazer hit', 'push', 'join', 'lock in', 'robot_start_executing', 'heal', 'energy gain']*

generate_entity_mapping ()
This method generates a dictionary containing the player id and the corresponding controlled entity.

Returns: dict of entity mapping

get_controlled_entities ()
This method returns the controlled entity dict.

Returns: dict which contains the mapping of entity to player

get_current_message ()
This method returns the latest (pop) game event message (JSON).

Returns: current message

topic = *'general'*

REST.RESTClient.**check_if_event_is_action** (msg)
This function checks if the msg contains an entityID.

Parameters: **msg** – current message
Returns: returns whether or not the event is an action with entity

REST.ResourceHandler module

class REST.ResourceHandler.**ResourceHandler** (api)
Bases: **object**
This class provides functions for interaction with the API resources.

add_resources ()
This function connects the API with the resources.

check_for_lobby_game (games**)**
This method waits until there are games in LOBBY state.

Parameters: **games** – all game ids that are currently active

create_consumer (game_id**)**
This method registers an consumer for hardware interaction.

Parameters: **game_id** – the given game identifier
Returns: Response from the server e.g. the pat and the id

get_all_robots (game_id**,** user_token**)**
This method returns all robot ids.

Parameters:
- **game_id** – the given game identifier
- **user_token** – the given consumer access token

Returns: all robot ids

get_controlled_entities (game_id**,** player_id**,** user_token**)**
This method returns the controlled entities of the given player.

Parameters:
- **user_token** – the given consumer access token
- **game_id** – the given game identifier
- **player_id** – the given player id

Returns: the id of the controlled robot from the given player

get_event_head (game_id**,** user_token**)**
This method returns the event head message from the API endpoint.

Parameters:
- **game_id** – the given game identifier
- **user_token** – the given consumer access token

Returns: the latest event

get_game_state (game_id**)**
This method returns the current state of the given game.

Parameters: **game_id** – the given game identifier
Returns: state of the game

get_games ()
This function returns all currently active games.

Returns: the game ids of all active games

get_player (game_id, player_id)
This function returns information about the given player in the given game.

Parameters:
- **player_id** – the given player id
- **game_id** – the given game identifier

Returns: information about the given player

get_players (game_id, user_token)
This function returns all active players in the given (game_id) game.

Parameters:
- **user_token** – the given consumer access token
- **game_id** – the given game identifier

Returns: the player ids of all active players

REST.Resources module

class REST.Resources.**ConsumersResource** (*args, **kwargs)
Bases: **simple_rest_client.resource.Resource**
This class represents all resources which represent the consumer endpoint.

actions = *{'create_consumer': {'method': 'POST', 'url': 'games/{}/consumers'}}*

class REST.Resources.**EventsResource** (*args, **kwargs)
Bases: **simple_rest_client.resource.Resource**
This class represents all resources which represent the event handling endpoint.

actions = *{'get_event_head': {'method': 'GET', 'url': 'games/{}/events/head?pat={}'}}*

class REST.Resources.**GamesResource** (*args, **kwargs)
Bases: **simple_rest_client.resource.Resource**
This class represents all resources which represent the games endpoint.

actions = *{'get_game_actions': {'method': 'GET', 'url': 'games/{}/actions'}, 'get_game_status': {'method': 'GET', 'url': 'games/{}/status'}, 'get_games': {'method': 'GET', 'url': 'games'}}*

class REST.Resources.**MapResource** (*args, **kwargs)
Bases: **simple_rest_client.resource.Resource**
This class represents all resources which represent the maps endpoint.

actions = *{'get_map': {'method': 'GET', 'url': 'games/{}/map'}}*

class REST.Resources.**PlayersResource** (*args, **kwargs)
Bases: **simple_rest_client.resource.Resource**
This class represents all resources which represent the players endpoint.

actions = *{'get_player': {'method': 'GET', 'url': 'games/{}/players/{}?pat={}'}, 'get_players': {'method': 'GET', 'url': 'games/{}/players?pat={}'}}*

class REST.Resources.**RobotsResource** (*args, **kwargs)
Bases: **simple_rest_client.resource.Resource**
This class represents all resources which represent the robots endpoint.

actions = *{'get_all_robots': {'method': 'GET', 'url': 'games/{}/entities/robots?pat={}'}, 'get_robot_info': {'method': 'GET', 'url': 'games/{}/entities/robots/{}/info'}, 'get_upgrades': {'method': 'GET', 'url': 'games/{}/entities/robots/{}/upgrades'}}*

Module contents

Indices and tables

- **genindex**
- **modindex**
- **search**

Index

wait_for_running_game()
(REST.ConnectionHandler.ConnectionHandler method)

Python Module Index

g

h

m

n

r