# NoSeMaze

***Release 1.0***

**May 12, 2022**

# CONTENTS:

CHAPTER

ONE

# NOSEMAZE CONTROLLER

## 1.1 NoSeMazeControl Modules

### 1.1.1 NoSeMaze Controller

This folder contains the UI of NoSeMaze Controller which control, save and load experiments in NoSeMaze setup.

### 1.1.2 Analysis package

#### Analysis

This package contains the modules used in analysis window. To implement new analysis, please change the analysis window implementation and designs accordingly.

#### Submodules

#### Analysis.Analysis module

This module contains methods used in analysis windows or in report in Email module.

Calculation of d' is left as commentar in weighted_binned_performance method.

Analysis.Analysis.**binned_performance**(*mouse: Models.Experiment.Mouse*, *bin_size: int*, *idx: int*) → list[float]

Calculate hit rate and correct rejection rate and returns them.

**Parameters**

- **mouse** (*Experiment.mouse class*) – Instance of Experiment.mouse class that is of interest.
- **bin_size** (*int*) – Size of bin.

  It is the number of trials used to calculate hit rate or correct rejection rate.
- **idx** (*int*) – Index of tab window selected on analysis window.

  It differentiate between left GNG or right GNG

**Returns** **binned_correct** – List of correct rate (hit and correct rejection altogether) to be shown in graphic view on analysis window.

**Return type** list of float

Analysis.Analysis.**n_trials_last_24**(*mouse: Models.Experiment.Mouse*) → int

Calculate number of trials in last 24h.

**Parameters** **mouse** (*Experiment.mouse class*) – Instance of mouse to calculate number of trials.

**Returns** **n_trials** – Number of trials since 1 day.

**Return type** int

Analysis.Analysis.**n_trials_performed**(*mouse: Models.Experiment.Mouse*) → int

Calculate number of trials performed by a mouse since the start of the experiment.

**Parameters** **mouse** (*Experiment.mouse class*) – Instance of mouse to calculate total trials performed.

**Returns** **total_trials** – Total trials performed since experiment start.

**Return type** int

Analysis.Analysis.**n_trials_since**(*mouse: Models.Experiment.Mouse*, *since: datetime.datetime*) → int

Calculate number of trials since a time.

**Parameters**

- **mouse** (*Experiment.mouse class*) – Instance of mouse to calculate number of trials of.
- **since** (*datetime.datetime*) – Instance of datetime.datetime as start point to calculate number of trials performed.

**Returns** **n_trials** – Number of trials since time given.

**Return type** int

Analysis.Analysis.**weighted_binned_performance**(*mouse: Models.Experiment.Mouse*, *bin_size: int*, *idx: int*) → tuple[list[float], list[float]]

In this function we examine performance by how many rewarded vs. unrewarded trials there were.

**Parameters**

- **mouse** (*Experiment.mouse class*) – Instance of Experiment.mouse class that is of interest.
- **bin_size** (*int*) – Size of bin.

  It is the number of trials used to calculate hit rate or correct rejection rate.
- **idx** (*int*) – Index of tab window selected on analysis window.

  It differentiate between left GNG or right GNG

**Returns**

- **binned_perf_p** (*list of float*) – List of hit rate in bin to be viewed in graphic view on analysis window.
- **binned_perf_m** (*list of float*) – List of correct rejection rate in bin to be viewed in graphic view on analysis window.

### Analysis.Conversion module

This module contains methods used for converting experiment data into matlab data. It is not used and not supported in NoSeMaze but still saved here for archive purpose.

Analysis.Conversion.**batch_convert**(*paths: list[str]*, *out_path: str*, *out_name: str*, *trial_parameter: int*, *verbose: bool = True*, *save_licks: bool = False*)

Convert files in batch.

**Parameters**

- **paths** (*list of str*) – List of paths to convert experiment data from.
- **out_path** (*str*) – Path to save the converted data.
- **out_name** (*str*) – Name of the saved file of the converted data.
- **trial_parameter** (*int*) – Parameter for output.
- **verbose** (*bool*) – Check if verbose mode should be used.
- **save_licks** (*bool*) – Check if licks should be saved.

Analysis.Conversion.**convert**()

Converting in a batch files with defined paths.

Analysis.Conversion.**load_experiment**(*path: str*) → tuple[Models.Experiment.Experiment, list[str], dict]

Read all experiment files in path and return the data.

**Parameters** **path** (*str*) – Path of the experiment files.

**Returns**

- **experiment** (*Experiment.Experiment*) – Last experiment loaded from .autmaus file.
- **data_files** (*list[str]*) – List of .mat files filename.
- **schedule_map** (*dict*) – Last schedule read from .csv file.

Analysis.Conversion.**read_schedule_map**(*path: str*) → dict

Read schedule map from csv file.

**Parameters** **path** (*str*) – Path of the csv file.

**Returns** **schedule_map** – Schedule map read from the csv file.

**Return type** dict

### Analysis.Performance module

This module contains method to show risk taking performance of mouses.

Analysis.Performance.**get_performance**(*all_data: list[dict[str, list[list[float]]]]*, *bins: int*) → list[list]

Get performance list in data.

**Parameters**

- **all_data** (*list of dict*) – Data is a list of data.
- **bins** (*int*) – Window to determine performance.

**Returns** **performance** – A list of performance with the length of data.

**Return type** 2d-list

Analysis.Performance.**load_data**(*filepath: str*, *start_date: datetime.datetime*, *stop_date: datetime.datetime*) → tuple[list[str], list[dict]]

Load experiment data.

**Parameters**

- **filepath** – Path to experiment file.
- **start_date** – Start date of time of interest.
- **stop_date** – Stop date of time of interest.

**Returns**

- **header** (*list*) – List of header
- **all_data** (*list of dict*) – List of dictionary of all data with headers as keys.

Analysis.Performance.**show_percent_correct**(*data: dict[str, list[list[float]]]*, *mouses: list[str]*, *bins: int*)

Plot rate of correct hit or correct rejection.

**Parameters**

- **data** (*2d-list of float*) – Data to be plotted.
- **mouse** (*str*) – Id of the mouse.
- **bins** (*int*) – Bin size.

Analysis.Performance.**show_percent_left_right**(*data: dict[str, list[list[float]]]*, *mouses: list[str]*, *bins: int*) → None

Plot rate of licks between left and right.

**Parameters**

- **data** (*2d-list of float*) – Data to be plotted.
- **mouse** (*str*) – Id of the mouse.
- **bins** (*int*) – Bin size.

Analysis.Performance.**show_risk_per_mouse**(*data: list[list[float]]*, *mouse: str*, *bins: int*) → None

Plot rate of risk taken per mouse.

**Parameters**

- **data** (*2d-list of float*) – Data to be plotted.
- **mouse** (*str*) – Id of the mouse.
- **bins** (*int*) – Bin size.

Analysis.Performance.**show_risk_per_odour**(*data: dict[str, list[list[float]]]*, *mouses: list[str]*, *bins: int*) → None

Plot rate of risk taken per odour.

**Parameters**

- **data** (*2d-list of float*) – Data to be plotted.
- **mouse** (*list of str*) – Id of the mouses
- **bins** (*int*) – Bin size.

Analysis.Performance.**show_total_risk**(*data: dict[str, list[list[float]]]*, *mouses: list[str]*, *bins: int*) → None

Plot rate of risk taken.

**Parameters**

- **data** (*2d-list of float*) – Data to be plotted.
- **mouse** (*list of str*) – Id of the mouse.
- **bins** (*int*) – Bin size.

### 1.1.3 Controllers package

#### Controllers

This package contains the controller of measurement thread and its worker.

#### Submodules

#### Controllers.ExperimentControl module

This module contains thread controller and thread.

As this module was developed rapidly per changing demand, some actually unused code might still be saved as comments and there may be some codes that is actually obsolete, but still not removed, as it might be used later on, if demand changed.

**class** Controllers.ExperimentControl.**ExperimentController**(*parent: PyQt5.QtWidgets.QMainWindow*)

Bases: object

**start**()

Slot for starting the experiment.

**stop**()

Slot for stopping experiment

**update_pref**(*new_pref: dict[str, object]*)

Slot for updating hardware preferences while the experiment is running

**Parameters** **new_pref** (*dict*) – The new hardware preferences should be used.

**class** Controllers.ExperimentControl.**ExperimentWorker**(*parent: Optional[PyQt5.QtCore.QObject] = None*)

Bases: PyQt5.QtCore.QObject

Worker of experiment thread. Sequence of thread is in trial method.

**animal_present**()

Checks whether animal is present in the port.

**check_licks**()

Check if a mouse has not licked in a period of time, then send a warning e-mail. Currently, all e-mail will be saved as text notification in Dropbox.

**check_status**()

Check if deadman's switch has be sent or not. Message will be sent at 8 AM, 5 PM, and 10 PM.

**finished**

Signal sent if an experiment is finished.

**Type** pyqtSignal

**get_present_animal**()

Returns the animal in the port. 'Rfid.check_rfid' method differs between NoSeMaze 1 and NoSeMaze 2.

**get_trial_daq**(*t: Optional[list] = None*, *odor_pulses: Optional[Union[object, list[float]]] = None*, *current_trial: Optional[Union[list, float]] = None*, *fv_pulse: Optional[float] = None*, *pretraining: bool = False*, *odour_training: bool = False*, *concatenate: bool = False*) → *daqface.DAQ.ThreadSafeAnalogInput* | *daqface.DAQ.DoAiMultiTaskOdourTraining* | *daqface.DAQ.DoAiConcatenatedWaitTrainingMultiTask* | *daqface.DAQ.DoAiConcatenatedPretrainingMultiTask* | *daqface.DAQ.DoAiMultiTask*

Initialise daq instance then returns it. Daq instance is responsible for data acquisition and the sequences of the actual trial. E.g. sequence of pretraining trial or wait training trial.

**Parameters**

- **t** (*list*) – Time axis.
- **odor_pulses** (*obj or list of float*) – List of odor parameters to be given to DAQ
- **current_trial** (*list of float*) – Current trial parameters. Details, see schedule generator.
- **fv_pulse** (*float*) – Duration of delay before final valve is opened.
- **pretraining** (*bool*) – Indicator if schedule is for pretraining.
- **odour_training** (*bool*) – Indicator if schedule is for odour training.
- **concatenate** (*bool*) – Indicator if schedule is for concatenated trial.

**Returns** **trial_daq** – Instance of daq class that is responsible of executing the trial sequence.

**Return type** daqface.daq-class

**reward**(*animal: Models.Experiment.Mouse*)

Initialise DAQ and execute giving water.

**Parameters** **animal** (*instance of Mouse class from Experiment module*) – Instance of mouse to be given the reward. Reward is given according to the schedule (Only amount of water. Reward is always given or probability is always 1).

**save_data**(*animal_id: str*, *timestamp: datetime.datetime*, *rewarded: list*, *wait_response: list*, *response: list*, *correct: bool*, *timeout: bool*, *pulses: numpy.array*, *time_axis: numpy.array*, *file_timestamp: datetime.datetime*, *lick_time: list[list]*, *licks: list[float]*, *wait_time: list[list]*)

Save experiment data. New data will be appended in the existing file. If there is no existing file or date is changed, a new file will be made.

**Parameters**

- **animal_id** (*str*) – Id of animal
- **timestamp** (*datetime.datetime*) – Timestamp of trial started.
- **rewarded** (*list*) – List of reward parameters (Reward probabilty and amount of water).
- **wait_respone** (*list*) – Response at wait window of left and right port. Currently, only left port is used.
- **response** (*list*) – Response at lick window of left and right port. Currently, only left port is used.

- **correct** (*bool*) – Indicator if trial is correctly performed (hit or correct rejection). Currently, correct is always True if water is given. That means, correct rejection will be falsely indicated as false.
- **timeout** (*bool*) – Indicator if a timeout is executed. Currently, there is no timeout.
- **pulses** (*ndarray*) – Array of pulses shown. Currently obsolete.
- **time_axis** (*ndarray*) – Array of indicator of time axis. Currently obsolete.
- **file_timestamp** (*datetime.datetime*) – Timestamp to be converted to file name.
- **lick_time** (*2d-list*) – List of timestamp of licks on left port and right port. Currently, only left port is used.
- **licks** (*list of float*) – List of number of licks on left port and right port. Currently, only left port is used.
- **wait_time** (*2d-list*) – List of timestamp of licks in wait window.

**timeout**()

Set the thread to sleep of a duration.

**trial**()

Sequence of trial. Contains pre-trial sequences, trial sequences in daq instance, and post-trial sequences.

**trial_end**

Signal sent if a trial has ended.

**Type** pyqtSignal

### 1.1.4 Designs package

#### Designs

This package contains the design classes of the windows used in the controller UI. It is recommended to seperate the implementation of a window and the design of the window into two separate scripts, that the scripts have a better overview.

The modules are created using PyQt5 UI generator and modified as needed.

#### Submodules

#### Designs.HardwareWindowEdited module

**class** Designs.HardwareWindowEdited.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### Designs.addWindow module

class Designs.addWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### Designs.adjustmentWidget module

class Designs.adjustmentWidget.**Ui_DockWidget**

Bases: object

**retranslateUi**(*DockWidget*)

**setupUi**(*DockWidget*)

### Designs.analysisWindow module

class Designs.analysisWindow.**GNGWidget**

Bases: PyQt5.QtWidgets.QWidget

class Designs.analysisWindow.**RiskWidget**

Bases: PyQt5.QtWidgets.QWidget

class Designs.analysisWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### Designs.animalWindow module

class Designs.animalWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### Designs.controlWindow module

class Designs.controlWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### Designs.controlWindowGraphicsView module

class Designs.controlWindowGraphicsView.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### Designs.hardwareWindow module

class Designs.hardwareWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### Designs.mailWindow module

class Designs.mailWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

#### Designs.mainWindow module

class Designs.mainWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

#### Designs.prefsWindow module

class Designs.prefsWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

#### Designs.settingWindow module

class Designs.settingWindow.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### 1.1.5 HelperFunctions package

#### HelperFunctions

This package contains useful methods used in the measurement thread.

#### Submodules

#### HelperFunctions.BeamCheck module

This module contains method to check if beam is broken, which signals the start of a trial.

HelperFunctions.BeamCheck.**check_beam**(*device*, *channels*, *beam_channel*)

Check if beam is broken. If beam is broken, TTL is high and data is measured at 5 V. If beam is not broken, TTL is low and data is measured at 0 V. If mean of data > 1, means that in 0.1 s there is a TTL high, ergo beam was broken.

**Parameters**

- **device** (*str*) – Name of device in NI-board where beam sensor is connected.
- **channels** (*int*) – Number of analog input channels read.
- **beam_channel** (*int*) – Index of beam channel.

**Returns** **broken** – Indicator if beam is broken or not.

**Return type** bool

### HelperFunctions.Email module

This module contains methods used for e-mailing. Currently, is used to write notification message in Dropbox. Codes to send e-mail is in commentar.

HelperFunctions.Email.**crash_error**(*exctype*, *value*, *tb*)

Get traceback if software is crashed and send e-mail. Currently, a message will be written in Dropbox.

**Parameters**

- **exctype** (*Type of Exception*) – Type of exception
- **value** (*int*) – Error id.
- **tb** (*obj*) – Traceback object.

HelperFunctions.Email.**deadmans_switch**(*experiment*)

Notification every day to indicate if experiment is running.

**Parameters** **exeperiment** (*Experiment.experiment*) – Instance of Experiment.experiment class to take information from.

HelperFunctions.Email.**parse_list**(*string*)

Method used to parse string of a list into a list of float, then sum it.

**Parameters** **string** (*str*) – String of a list to be parsed.

**Returns** **result** – Sum of the parsed list.

**Return type** float

HelperFunctions.Email.**send**(*subject*, *content*, *attachment*)

Write a message with subject, content and attachment in Dropbox.

**Parameters**

- **subject** (*str*) – Subject of message
- **content** (*str*) – Content of message
- **attachement** (*list*) – List of attachment.

HelperFunctions.Email.**warning_licks**(*logs_path*, *namelist*)

Send warning e-mail if any mouse has not licked in a period of time. Currently, a message will be written in Dropbox.

**Parameters**

- **logs_path** (*str*) – Directory path to folder with all licks log of mouses.
- **namelist** (*list*) – List of mouse's ID which has not licked in a period of time.

### HelperFunctions.Filter module

This module is used to filter analog data read from NI-Board.

HelperFunctions.Filter.**Gauss_Filter**(*data*, *k*, *fs*, *fcut*)

HelperFunctions.Filter.**Square_Filter**(*data*, *fs*, *fcut*)

### HelperFunctions.RFID module

This module read the rfid tag of the mouse present at the port.

HelperFunctions.RFID.**check_rfid**(*port*, *n*)

Read the rfid from the rfid reader with a 2 seconds timeout. If no valid rfid read in 2 seconds, returns 'default'.

**Parameters**

- **port** (*str*) – COM-port of rfid reader.
- **n** (*int*) – Total number of ID. In this case, it is not used.

**Returns** **out** – ID of mouse present.

**Return type** str

### HelperFunctions.Reward module

This module contains methods to deliver reward at different situations.

HelperFunctions.Reward.**deliver_reward**(*do_device*, *samp_rate*, *secs*, *sync_clock*, *ai_channels*)

Deliver reward to a device.

**Parameters**

- **do_device** (*str*) – Name of digital out device
- **samp_rate** (*int*) – Sample rate specified in hardware configuration.
- **secs** (*float*) – Duration that the ventil should be opened (water amount).
- **sync_clock** (*str*) – Clock preferred for synchronised reading.
- **ai_channels** (*int*) – Number of analog input channels read.

**Returns** **out** – Currently, there is no analog data read.

**Return type** None

HelperFunctions.Reward.**deliver_reward_static**(*do_device*, *secs*)

Deliver reward from a ventil.

**Parameters**

- **do_device** (*str*) – Name of device.
- **secs** (*float*) – Duration that the ventil should be opened (water amount).

**Returns** **out** – Currently no analog input is read.

**Return type** None

HelperFunctions.Reward.**deliver_reward_static_concatenate**(*device1*, *device2*, *secs*, *index*)

Deliver reward for concetanted training.

**Parameters**

- **device1** (*str*) – Name of device 1 (left port).
- **device2** (*str*) – Name of device 2 (right port).
- **secs** (*float*) – Duration that the ventil should be opened (water amount).
- **index** (*int*) – Index of odour actually presented.

**Returns** **out** – There is currently no analog input read.

**Return type** None

HelperFunctions.Reward.**deliver_reward_static_two**(*device1*, *device2*, *secs1*, *secs2*)

Deliver reward for concetanted training.

**Parameters**

- **device1** (*str*) – Name of device 1 (left port).
- **device2** (*str*) – Name of device 2 (right port).
- **secs1** (*float*) – Duration that left ventil should be opened (water amount).
- **secs2** (*float*) – Duration that right ventil should be opened (water amount).

**Returns** **out** – There is currently no analog input read.

**Return type** None

### 1.1.6 PyPulse package

#### PyPulse

A set of tools for generating complex pulse trains for digital inputs, largely for use in driving high speed valves / PWM.

This contains a slightly different methods that from PyPulse in schedule generator because of pulse parameters that is needed to be prepared for DAQ.

### Submodules

### PyPulse.PulseGeneration module

This module contains methods to generate pulse for use in DAQ.

PyPulse.PulseGeneration.**concatenated_pulse**(*sampling_rate*, *params*, *number*, *total*)

PyPulse.PulseGeneration.**dummy_noise_pulse**(*sampling_rate*, *params*)

PyPulse.PulseGeneration.**extended_square_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*)

PyPulse.PulseGeneration.**fv_pulse**(*sampling_rate*, *params_list*, *length*, *fv_delay*)

PyPulse.PulseGeneration.**lick_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*)

PyPulse.PulseGeneration.**multi_noise_pulse**(*sampling_rate*, *global_onset*, *global_offset*, *params_list*)

PyPulse.PulseGeneration.**multi_simple_pulse**(*sampling_rate*, *global_onset*, *global_offset*, *params_list*)

PyPulse.PulseGeneration.**noise_pulse**(*sampling_rate*, *params*)

PyPulse.PulseGeneration.**plume_pulse**(*sampling_rate*, *params*)

PyPulse.PulseGeneration.**random_shatter_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*, *shatter_frequency*, *target_duty*, *amp_min*, *amp_max*, *extend=False*)

this function generates a shattered pulse based on major pulse frequency and duty, as well as shatter frequency. The function will generate standard pulse and then shatter it, with the duty of each shattered pulse randomised. The function will aim to keep the integral of the pulse at duty * target duty

PyPulse.PulseGeneration.**random_simple_pulse**(*sampling_rate*, *params*)

PyPulse.PulseGeneration.**shatter_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*, *shatter_frequency*, *shatter_duty*)

PyPulse.PulseGeneration.**simple_pulse**(*sampling_rate*, *params*)

PyPulse.PulseGeneration.**simple_pulse_static**(*sampling_rate*, *params*)

PyPulse.PulseGeneration.**spec_time_pulse**(*sampling_rate*, *params*)

PyPulse.PulseGeneration.**square_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*)

### PyPulse.PulseInterface module

This module contains method to input pulse parameters to get desired pulse.

PyPulse.PulseInterface.**make_pulse**(*sampling_rate*, *global_onset*, *global_offset*, *params_list*)

**Parameters**

- **sampling_rate** (*int*) – Sampling rate defined in hardware configuration
- **global_onset** (*float*) – Number of element set to be an onset.

- **global_offset** (*float*) – Number of element set to be an offset.
- **params_list** (*list*) – List of pulse parameters.

**Returns**

- **pulse_matrix** (*2d-list*) – List of pulse.
- **t** (*ndarray*) – Time axis.

### 1.1.7 TrialLogic package

#### TrialLogic

This package contains modules for deciding trial results.

#### Submodules

#### TrialLogic.TrialConditions module

This module contains methods to analyse read data.

**class** TrialLogic.TrialConditions.**TrialResult**(*value*)

Bases: enum.Enum

An enumeration.

**correct_rejection = 2**

**correct_response = 1**

**false_alarm = 4**

**miss = 3**

TrialLogic.TrialConditions.**lick_detect**(*lick_data*, *threshold*, *percent_accepted*)

Detect lick and analyse it if it is considered licked or not.

**Parameters**

- **lick_data** (*ndarray*) – Read data to be analysed.
- **threshold** (*int*) – Threshold if data is considered TTL high.
- **percent_accepted** (*float*) – Ratio of TTL high in data to be considered licked.

**Returns** **response** – Response is true, if number of licks is greater than 2 or if the ratio is greater than indicated.

**Return type** bool

TrialLogic.TrialConditions.**licks_number**(*lick_data*, *threshold*, *samp_rate*, *start_time*)

Get the number of licks from data.

**Parameters**

- **lick_data** (*ndarray*) – Read data to be analysed.
- **threshold** (*int*) – Threshold if data is considered TTL high.
- **samp_rate** (*int*) – Sample rate definded in hardware configuration.

- **start_time** (*datetime.datetime*) – Start time of a trial.

**Returns**

- **lick_times** (*list*) – Timestamps of licks.
- **licks** (*int*) – Number of licks.

TrialLogic.TrialConditions.**trial_result**(*_rewarded*, *_response_l*, *_response_r*)

Analyse trial, if it is correct.

**Parameters**

- **_rewarded** (*list*) – List of reward parameteres (reward probability and amount of reward).
- **_response_l** (*bool*) – Indicator if left port is licked.
- **_response_r** (*bool*) – Indicator if right port is licked.

**Returns**

- **TrialResult_enum** (*TrialResult*) – Enum of trial result.

  4 indicators - correct response, correct rejection, miss, and false alarm.
- **correct** (*bool*) – Indicator if correct response or correct rejection.
- **timeout** (*bool*) – Indicator if timeout is executed.

### 1.1.8 Windows package

This module contains all implementation of windows in UI. There are adjustment window, hardware window, control window, animal window, e-mail window and analysis window.

#### Submodules

#### Windows.AppWindows module

This module contains all implementation of windows in UI. There are adjustment window, hardware window, control window, animal window, e-mail window and analysis window.

**class** Windows.AppWindows.**AdjustmentWidget**(*cam*, *settings*, *parent=None*)

Bases: PyQt5.QtWidgets.QDockWidget, *Designs.adjustmentWidget.Ui_DockWidget*

**set_brightness**(*value*)

**set_contrast**(*value*)

**set_saturation**(*value*)

**class** Windows.AppWindows.**AnalysisWindow**(*experiment*, *parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, *Designs.analysisWindow.Ui_MainWindow*

**current_animal**()

**display_animal_performance**()

**display_group_performance**()

**on_animal_selected**()

**on_tab_selected**(*index*)

**populate_stats_table**()

**class** Windows.AppWindows.**AnimalWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, *Designs.animalWindow.Ui_MainWindow*

**add_row**()

**add_schedule**()

**animal_selected**()

**change_false**()

**change_true**()

**changed**

int = ..., arguments: Sequence = ...) -> PYQT_SIGNAL

types is normally a sequence of individual types. Each type is either a type object or a string that is the name of a C++ type. Alternatively each type could itself be a sequence of types each describing a different overloaded signal. name is the optional C++ name of the signal. If it is not specified then the name of the class attribute that is bound to the signal is used. revision is the optional revision of the signal that is exported to QML. If it is not specified then 0 is used. arguments is the optional sequence of the names of the signal's arguments.

**Type** pyqtSignal(*types, name

**Type** str = ..., revision

**closeEvent**(*self*, *QCloseEvent*)

**current_animal**()

**current_sched_index**()

**populate_animal_table**(*animal_list*)

**remove_row**()

**remove_schedule**()

**saved**

int = ..., arguments: Sequence = ...) -> PYQT_SIGNAL

types is normally a sequence of individual types. Each type is either a type object or a string that is the name of a C++ type. Alternatively each type could itself be a sequence of types each describing a different overloaded signal. name is the optional C++ name of the signal. If it is not specified then the name of the class attribute that is bound to the signal is used. revision is the optional revision of the signal that is exported to QML. If it is not specified then 0 is used. arguments is the optional sequence of the names of the signal's arguments.

**Type** pyqtSignal(*types, name

**Type** str = ..., revision

**schedule_selected**()

**update_animals**()

class Windows.AppWindows.**ControlWindow**(*experiment*, *parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, *Designs.controlWindow.Ui_MainWindow*

**adjust_cam1**()

**adjust_cam2**()

**closeEvent**(*self*, *QCloseEvent*)

**configure_settings**(*settings*)

**get_cam1**()

**get_cam2**()

**get_res_cam1**()

**get_res_cam2**()

**on_animal_selected**()

**populate_table**()

**qsize_to_string**(*qsize*)

**set_cam1**(*cam*)

**set_cam2**(*cam*)

**set_res_cam1**(*x*, *y*)

**set_res_cam2**(*x*, *y*)

**update_licks_view**(*animal*)

class Windows.AppWindows.**HardwareWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, *Designs.hardwareWindow.Ui_MainWindow*

**change**()

**closeEvent**(*self*, *QCloseEvent*)

**new_pref**

int = ..., arguments: Sequence = ...) -> PYQT_SIGNAL

types is normally a sequence of individual types. Each type is either a type object or a string that is the name of a C++ type. Alternatively each type could itself be a sequence of types each describing a different overloaded signal. name is the optional C++ name of the signal. If it is not specified then the name of the class attribute that is bound to the signal is used. revision is the optional revision of the signal that is exported to QML. If it is not specified then 0 is used. arguments is the optional sequence of the names of the signal's arguments.

**Type** pyqtSignal(*types, name

**Type** str = ..., revision

**save_preferences**()

**saved**

int = ..., arguments: Sequence = ...) -> PYQT_SIGNAL

types is normally a sequence of individual types. Each type is either a type object or a string that is the name of a C++ type. Alternatively each type could itself be a sequence of types each describing a different overloaded signal. name is the optional C++ name of the signal. If it is not specified then the name of the class attribute that is bound to the signal is used. revision is the optional revision of the signal that is exported to QML. If it is not specified then 0 is used. arguments is the optional sequence of the names of the signal's arguments.

**Type** pyqtSignal(*types, name

**Type** str = ..., revision

**saved_status**()

**set_preferences**(*prefs*)

**zero_edit**()

**class** Windows.AppWindows.**MailWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, *Designs.mailWindow.Ui_MainWindow*

**add_address**(*new_addrs*)

**get_address**()

**populate_mailing_list**(*mlist*)

**remove_address**()

**save**(*mlist*)

**class** Windows.AppWindows.**PreferencesWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, *Designs.prefsWindow.Ui_MainWindow*

**save_preferences**()

**select_save_path**()

**set_preferences**(*prefs*)

### 1.1.9 daqface package

#### daqface

A high-level python interface for a number of common NI DAQmx hardware tasks. Some unused classes are removed from the original version as to maintain overview and avoid confusion.

### Submodules

### daqface.DAQ module

class daqface.DAQ.**DoAiConcatenatedPretrainingMultiTask**(*ai_device: str*, *ai_channels: str*, *do_device: str*, *fv_device: str*, *reward_device_l: str*, *reward_device_r: str*, *samp_rate: int*, *secs: float*, *odor_write: float*, *fv_write: float*, *sync_clock: str*, *static: bool*, *lick_channel_l: int*, *lick_channel_r: int*, *beam_channel: int*, *rewarded: list[object]*)

Bases: object

DAQ for concatenated pretraining schedule. Not used in current implementation.

**CheckLicks**(*analogData: numpy.ndarray*, *idx: int*)

Check if reward should be given. In this case, reward is always given regardless of performance.

**Parameters**

- **analogData** (*ndarray*) – Measured data of time window of interest.
- **idx** (*int*) – Index of odour that is actually presented. E.g., for 1st odour is idx = 0.

**Returns** **water_given** – List of indicator when water is given between 3 odours at left port and right port.

**Return type** 2d-list

**ClearTasks**()

Stop task and clear it

**DoTask**()

Start task and execute sequences.

**Returns**

- **analogData** (*ndarray*) – All measured data read during the trial
- **waitData** (*ndarray*) – Measured data read during the wait window
- **shouldLickData** (*ndarray*) – Measured data read during the lick window
- **timestamp1** (*datetime.datetime*) – Timestamp of the start of odour presentation
- **timestamp2** (*datetime.datetime*) – Timestamp of the start of trial.
- **water_given** (*list*) – List of indicator if water are given on left port or right port.

class daqface.DAQ.**DoAiConcatenatedWaitTrainingMultiTask**(*ai_device: str*, *ai_channels: int*, *do_device: str*, *fv_device: str*, *reward_device_l: str*, *reward_device_r: str*, *samp_rate: int*, *secs: float*, *odor_write: float*, *fv_write: float*, *sync_clock: str*, *static: bool*, *lick_channel_l: int*, *lick_channel_r: int*, *beam_channel: int*, *rewarded: list[object]*)

Bases: object

DAQ for concatenated wait training schedule. Not used in current implementation.

**CheckLicks**(*analogData: numpy.ndarray*, *idx: int*)

Check if reward should be given.

**Parameters**

- **analogData** (*ndarray*) – Measured data of time window of interest.
- **idx** (*int*) – Index of odour that is actually presented. E.g., for 1st odour is idx = 0.

**Returns** **water_given** – List of indicator when water is given between 3 odours at left port and/or right port.

**Return type** 2d-list

**ClearTasks**()

Stop task and clear it

**DoTask**()

Start task and execute sequences.

**Returns**

- **analogData** (*ndarray*) – All measured data read during the trial
- **waitData** (*ndarray*) – Measured data read during the wait window
- **shouldLickData** (*ndarray*) – Measured data read durign the lick window
- **timestamp1** (*datetime.datetime*) – Timestamp of the start of odour presentation
- **timestamp2** (*datetime.datetime*) – Timestamp of the start of trial.
- **water_given** (*list*) – List of indicator if water are given on left port or right port.

class daqface.DAQ.**DoAiMultiTask**(*ai_device: str*, *ai_channels: int*, *do_device: str*, *fv_device: str*, *reward_device_l: str*, *reward_device_r: str*, *samp_rate: int*, *secs: float*, *odor_write: list[object]*, *fv_write: numpy.ndarray*, *sync_clock: str*, *static: bool*, *wait_delay: float*, *lick_delay: float*, *lick_channel_l: int*, *lick_channel_r: int*, *beam_channel: int*, *rewarded: list[float]*)

Bases: object

DAQ for normal GNG or risk trial

**CheckLicks**()

Check if reward should be given.

**Returns** **water_given** – List of indicator if water is given at left port and/or right port.

**Return type** list

**ClearTasks**()

Stop task and clear it.

**DoTask**()

Start task and execute sequences.

**Returns**

- **analogData** (*ndarray*) – All measured data read during the trial
- **waitData** (*ndarray*) – Measured data read during the wait window
- **shouldLickData** (*ndarray*) – Measured data read durign the lick window
- **timestamp1** (*datetime.datetime*) – Timestamp of the start of odour presentation

- **timestamp2** (*datetime.datetime*) – Timestamp of the start of trial.
- **water_given** (*list*) – List of indicator if water are given on left port or right port.

class daqface.DAQ.**DoAiMultiTaskOdourTraining**(*ai_device*, *ai_channels*, *do_device*, *fv_device*, *reward_device_l*, *reward_device_r*, *samp_rate*, *secs*, *odor_write*, *fv_write*, *sync_clock*, *static*, *thorax_delay*, *lick_delay*, *lick_channel_l*, *lick_channel_r*, *beam_channel*, *rewarded*)

Bases: object

DAQ for normal GNG odor association training. That is reward is always given after odor presentation regardless of performance.

**CheckLicks**()

Check how reward should be presented. Reward is always given after odor presentation regardless of performance.

**Returns**

- **licks_l** (*int*) – Number of licks in lick window at left port during trial.
- **licks_r** (*int*) – Number of licks in lick window at right port during trial.
- **timestamps_l** (*list*) – Timestamps of licks in lick window.
- **timestamps_r** (*list*) – Timestamps of licks in lick window.
- **water_given** (*list*) – List of indicator if water is given at left port and/or right port.

**ClearTasks**()

Stop task then clear it

**DoTask**()

Start task and execute sequences.

**Returns**

- **analogData** (*ndarray*) – All measured data read during the trial
- **waitData** (*ndarray*) – Measured data read during the wait window
- **shouldLickData** (*ndarray*) – Measured data read durign the lick window
- **timestamp_1** (*datetime.datetime*) – Timestamp of the start of odour presentation
- **water_given** (*list*) – List of indicator if water are given on left port or right port.
- **licks** (*list of int*) – List of number of licks at left port and right port.
- **timestamps** (*2d-list*) – List of timestamps of licks at left port and right port.
- **timestamp_2** (*datetime.datetime*) – Timestamp of the start of trial.

class daqface.DAQ.**DoAiMultiTaskWaitTraining**(*ai_device*, *ai_channels*, *do_device*, *fv_device*, *reward_device_l*, *reward_device_r*, *samp_rate*, *secs*, *odor_write*, *fv_write*, *sync_clock*, *static*, *thorax_delay*, *lick_delay*, *lick_channel_l*, *lick_channel_r*, *beam_channel*, *rewarded*)

Bases: object

DAQ for strict GNG wait training. That means, reward will not be given if mouse does not wait. Not used in current implementation.

**CheckLicks**()

Check if reward should be given. If mouse does not wait and licked in wait window, no reward will be given.

**Returns**

- **licks_l** (*int*) – Number of licks in lick window at left port during trial.
- **licks_r** (*int*) – Number of licks in lick window at right port during trial.
- **timestamps_l** (*list*) – Timestamps of licks in lick window.
- **timestamps_r** (*list*) – Timestamps of licks in lick window.
- **water_given** (*list*) – List of indicator if water is given at left port and/or right port.

**ClearTasks**()

**DoTask**()

Start task and execute sequences.

**Returns**

- **analogData** (*ndarray*) – All measured data read during the trial
- **waitData** (*ndarray*) – Measured data read during the wait window
- **shouldLickData** (*ndarray*) – Measured data read durign the lick window
- **timestamp_1** (*datetime.datetime*) – Timestamp of the start of odour presentation
- **water_given** (*list*) – List of indicator if water are given on left port or right port.
- **licks** (*list of int*) – List of number of licks at left port and right port.
- **timestamps** (*2d-list*) – List of timestamps of licks at left port and right port.
- **timestamp_2** (*datetime.datetime*) – Timestamp of the start of trial.

**class** daqface.DAQ.**NiUsbDigitalOut**(*device: str*, *secs: float*, *on: int*)

Bases: object

Send digital signal out to one device.

**ClearTasks**()

Stop task and clear it

**DoTask**()

Start task and execute sequence

**class** daqface.DAQ.**NiUsbDigitalOutTwoDevices**(*device1: str*, *device2: str*, *secs1: float*, *secs2: float*, *on1: int*, *on2: int*)

Bases: object

Send digital signals out to two devices simultaneously. For giving reward.

**ClearTask**()

Stop task and clear it.

**DoTask**()

**class** daqface.DAQ.**NiUsbDigitalOutTwoDevicesC**(*device1: str*, *device2: str*, *secs: float*, *on1: int*, *on2: int*)

Bases: object

Send digital signal out to two channels simultaneously for concatenated training. It is currently not used

**ClearTask**()

Stop and clear all tasks.

**DoTask**()

Start task and execute sequences

class daqface.DAQ.**ThreadSafeAnalogInput**(*ai_device: str*, *channels: int*, *samp_rate: int*, *secs: float*, *clock: str = ''*)

Bases: object

Read thread safe analog input.

**ClearTasks**()

Stop task and clear it

**DoTask**()

Start task and read input.

**Returns** **analogData** – Data measured.

**Return type** ndarray

class daqface.DAQ.**ThreadSafeDigitalOut**(*device: str*, *samprate: int*, *secs: float*, *write: numpy.ndarray*, *clock: str*, *ai_channels: int*)

Bases: object

Send thread safe digital signal out to a device.

**ClearTasks**()

Stop task and clear it

**DoTask**()

Start task and execute sequence

### daqface.Utils module

daqface.Utils.**binary_to_digital_map**(*bin_buffer: numpy.ndarray*) → numpy.ndarray

If we have a sequence of 1s and 0s corresponding to digital ON and OFF times, we want to map this to digital commands that the digital ports can read and implement.

Input should be mapped as a 2d numpy array (dtype=numpy.uint32), rows are individual lines, columns are continuous time points. Ignores sampling rate until we tell NIDAQmx what it is.

**Parameters** **bin_buffer** (*ndarray*) – An integer array of zeros and ones to be converted.

**Returns** **digital** – Mapped digital commands.

**Return type** ndarray

### 1.1.10 main module

**class** main.**MainApp**

Bases: PyQt5.QtWidgets.QMainWindow, *Designs.mainWindow.Ui_MainWindow*

The main window class of the UI.

Inherits QMainWindow from QtWidgest and uses UI design from mainWindow.py.

**saved_status**

Status if experiments already saved. Checked while closing the windows and changed if any trials are executed or experiment is saved.

**Type** bool

**config_path**

Path to the config files.

**Type** str

**hardware_prefs**

Hardware preferences.

**Type** dict

**dropbox_path**

Path to dropbox to save deadman-switch, warning and error messages.

**Type** str

**hardware_window**

Hardware window to configure hardware preferences.

**Type** instance of AppWindows.HardwareWindow class

**animal_window**

Animal window to input animals in the experiment and their schedules.

**Type** instance of AppWindows.AnimalWindow class

**analysis_window**

Analysis window that shows the performance curve of the experiment and of each mouses respectively.

**Type** instance of AppWindows.AnalysisWindow class

**mail_window**

Mail window to input the mailing list.

**Type** instance of AppWindows.MailWindow class

**control_window**

Control window which shows video feed of usb cameras connected. Currently not maintained.

**Type** instance of AppWindows.ControlWindow class

---

**Note:** E-Mailing functionality is deprecated as messages are now saved in a cloud folder.

---

closeEvent(*event*)

Things to be executed if closeEvent occured (x in main window is clicked).

**Parameters** **event** (*instance of QCloseEvent*) – Events with close event information sent to the window.

experiment_saved()

Changing status and name of window if experiment is saved.

load_config_data()

Load hardware configuration data. Load files from path defined in config_path.

**Returns** **hardware_configs** – Hardware configurations saved in hardware.config

**Return type** dict

load_dropbox_path()

Load dropbox path saved in dropbox.txt to dropbox_path_file variable.

**Returns** **dropbox_path_file** – Path to dropbox file for messages to be written.

**Return type** str

load_experiment()

Load experiment.

loaded

pyqtSignal sent if experiment is loaded.

**Type** QtCore.pyqtSignal

on_trial_selected()

Update trial if trial is selected.

open_analysis_window()

Open analysis window.

open_animal_window()

Open animal window.

open_control_window()

Open control window.

open_hardware_window()

Open hardware window.

open_mail_window()

Open mail window.

save_experiment()

Save experiment.

saved

pyqtSignal sent if experiment is saved.

**Type** QtCore.pyqtSignal

set_dropbox_path()

Set dropbox path in which the deadman-switch, warning and error messages to be saved.

**setup_experiment_bindings**(*experiment*)

Set up experiment in the GUI and populate the experiment table in the GUI.

**experiment** [instance of Experiment.Experiment class] Experiment to be set up.

**show_about**()

Show the *about* message.

**status_changed**()

Changing status and name of window if a trial is executed.

**thread_control**()

Stop experiment control thread.

**update_data_view**()

Update data view of the trial.

**update_experiment_info**()

Update experiment labels in the GUI.

**update_graphics_view**(*trial*)

Update graph view to the odour signal of the selected trial.

**Parameters** **trial** (*int*) – Trial index to be viewed.

**update_trial_view**()

Update graph view of the trial if a trial is executed.

**windows_control**()

Check if any window is opened, then close it.

main.**main**()

Main method to be called.

main.**my_exception_hook**(*exctype*, *value*, *traceback*)

Custom exception hook.

**Parameters**

- **exctype** (*type of BaseException*) – Type of exception.
- **value** (*BaseException*) – Exception occured.
- **traceback** (*TracebackType*) – Traceback up to the exception.

CHAPTER

TWO

# NOSEMAZE SCHEDULE

## 2.1 NoSeMazeSchedule Modules

### 2.1.1 Schedule Generator

A GUI tool for generating schedules - sequences of trials that run in NoSeMaze

Python 3.10+ (on Windows, WinPython is recommended). GUI is PyQt5/pyqtgraph based. Requires pypulse to run.

### 2.1.2 Exceptions module

Custom Exceptions Used In Autonomouse Schedule

**exception** Exceptions.**RewardMapError**

Bases: Exception

### 2.1.3 Generation package

#### Generation

This package contains the Gen module. The module is used to generate randomize sequence of trials.

#### Submodules

#### Generation.Gen module

This module contains methods to generate randomise sequence of trials.

Generation.Gen.**odor_sequence**(*odour_choice*, *n_trials*)

Randomized sequence after odour.

**Parameters**

- **odour_choice** (*list*) – List of odours to be randomised from.
- **n_trials** (*int*) – Number of trials in a schedule defined in UI.

Generation.Gen.**reward_sequence**(*n_trials*)

Generate sequence after reward.

**n_trials** [int] Number of trials in a schedule defined in UI.

### 2.1.4 ScheduleDesigns package

#### ScheduleDesigns

This package contains the UI desings for the schedule widgets and some other widgets. It is recommended to separate implementation script and design script and put the design script here.

The modules are created using PyQt5 UI generator and modified as needed.

#### Submodules

#### ScheduleDesigns.NoSeMazeConcatenatedScheduleDesign module

**class** ScheduleDesigns.NoSeMazeConcatenatedScheduleDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

#### ScheduleDesigns.NoSeMazeScheduleDesign module

**class** ScheduleDesigns.NoSeMazeScheduleDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

#### ScheduleDesigns.concGNGDesign module

**class** ScheduleDesigns.concGNGDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.contCorrDesign module

class ScheduleDesigns.contCorrDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.corrDesign module

class ScheduleDesigns.corrDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.corrDifficultySwitchCameraTriggerDesign module

class ScheduleDesigns.corrDifficultySwitchCameraTriggerDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.corrDifficultySwitchDesign module

class ScheduleDesigns.corrDifficultySwitchDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.corrOnsetDisruptDesign module

class ScheduleDesigns.corrOnsetDisruptDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.corrRandomFrequency2Design module

class ScheduleDesigns.corrRandomFrequency2Design.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.corrRandomFrequencyDesign module

class ScheduleDesigns.corrRandomFrequencyDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.mainDesign module

class ScheduleDesigns.mainDesign.**Ui_MainWindow**

Bases: object

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### ScheduleDesigns.pretrainDesign module

class ScheduleDesigns.pretrainDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.scheduleBeastDesign module

### ScheduleDesigns.shatterValveTestDesign module

class ScheduleDesigns.shatterValveTestDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.simpleCorrDesign module

class ScheduleDesigns.simpleCorrDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### ScheduleDesigns.simpleGNGDesign module

class ScheduleDesigns.simpleGNGDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

#### ScheduleDesigns.valveMapDesign module

class ScheduleDesigns.valveMapDesign.**Ui_Form**

Bases: object

**retranslateUi**(*Form*)

**setupUi**(*Form*)

### 2.1.5 ScheduleModels package

#### ScheduleModels

Implementation of the actual schedule generation can be found in *ScheduleWidgets.py*. To add another trial sequence, implement a new schedule widgets in *ScheduleWidgets.py*. It will be automatically listed in the drop down list. It is recommended to separate the implementation script and the design script and put the implementation script in the module *ScheduleWidgets.py*

#### Submodules

#### ScheduleModels.ScheduleView module

This module contains the model of the table in Schedule Generator UI.

class ScheduleModels.ScheduleView.**ScheduleModel**(*headerdata*, *arraydata*, *parent=None*, **args*)

Bases: PyQt5.QtCore.QAbstractTableModel

**columnCount**(*self*, *parent: QModelIndex = QModelIndex()*) → int

**data**(*self*, *QModelIndex*, *role: int = Qt.ItemDataRole.DisplayRole*) → Any

**headerData**(*self*, *int*, *Qt.Orientation*, *role: int = Qt.ItemDataRole.DisplayRole*) → Any

**rowCount**(*self*, *parent: QModelIndex = QModelIndex()*) → int

#### ScheduleModels.ScheduleWidgets module

This module contains the implementation of the schedule widgets. If a new trial sequence is needed, then a new schedule widget is also needed.

Methods needed in a class implementation of a schedule widget is: generate_schedule() and pulse_parameters().

class ScheduleModels.ScheduleWidgets.**NoSeMazeScheduleWidget**(*parentUi=None*)

Bases: PyQt5.QtWidgets.QWidget, *ScheduleDesigns.NoSeMazeScheduleDesign.Ui_Form*

**change_reward_map**()

**flatten_value**(*value*)

**generate_schedule**(*valence_map*)

**pulse_parameters**(*trial*)

#### ScheduleModels.Widgets module

This module contains the ValveMapWidget, that is used in the UI.

**class** ScheduleModels.Widgets.**ValveMapWidget**(*parentUi=None*)

Bases: PyQt5.QtWidgets.QWidget, *ScheduleDesigns.valveMapDesign.Ui_Form*

**change_valve_map**()

**flatten_value**(*value*)

**get_valence_map**()

### 2.1.6 SchedulePyPulse package

#### SchedulePyPulse

A set of tools for generating complex pulse trains for digital inputs, largely for use in driving high speed valves / PWM. PyPulse in NoSeMazeController contains some modification from this PyPulse.

#### Submodules

#### SchedulePyPulse.PulseGeneration module

This module contains methods used to make pulses after PulseInterface decide, which pulse is to be made.

SchedulePyPulse.PulseGeneration.**concatenated_pulse**(*sampling_rate*, *params*, *number*, *total*)

SchedulePyPulse.PulseGeneration.**dummy_noise_pulse**(*sampling_rate*, *params*)

SchedulePyPulse.PulseGeneration.**extended_square_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*)

SchedulePyPulse.PulseGeneration.**multi_noise_pulse**(*sampling_rate*, *global_onset*, *global_offset*, *params_list*)

SchedulePyPulse.PulseGeneration.**multi_simple_pulse**(*sampling_rate*, *global_onset*, *global_offset*, *params_list*)

SchedulePyPulse.PulseGeneration.**noise_pulse**(*sampling_rate*, *params*)

SchedulePyPulse.PulseGeneration.**plume_pulse**(*sampling_rate*, *params*)

SchedulePyPulse.PulseGeneration.**random_shatter_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*, *shatter_frequency*, *target_duty*, *amp_min*, *amp_max*, *extend=False*)

This function generates a shattered pulse based on major pulse frequency and duty, as well as shatter frequency. The function will generate standard pulse and then shatter it, with the duty of each shattered pulse randomised. The function will aim to keep the integral of the pulse at duty * target duty

SchedulePyPulse.PulseGeneration.**random_simple_pulse**(*sampling_rate*, *params*)

SchedulePyPulse.PulseGeneration.**shatter_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*, *shatter_frequency*, *shatter_duty*)

SchedulePyPulse.PulseGeneration.**simple_pulse**(*sampling_rate*, *params*)

SchedulePyPulse.PulseGeneration.**spec_time_pulse**(*sampling_rate*, *params*)

SchedulePyPulse.PulseGeneration.**square_pulse**(*sampling_rate*, *duration*, *frequency*, *duty*)

#### SchedulePyPulse.PulseInterface module

This module contains method to make pulse according to parameter inputed.

SchedulePyPulse.PulseInterface.**make_pulse**(*sampling_rate*, *global_onset*, *global_offset*, *params_list*)

### 2.1.7 scheduleMain module

**class** scheduleMain.**MainApp**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, *ScheduleDesigns.mainDesign.Ui_MainWindow*

The MainApp of schedule generator UI. Inherits QMainWindow and uses design from mainDesign.

**parent**

Parent of the main window.

**Type** QtWidgets

**current_schedule_type**

**Type** widget from ScheduleWidgets

**draw_pulse**()

Slot for drawing or redrawing pulse in UI if a new trial is selected.

**generate**()

Slot for generating schedule if generate button is clicked.

**open_user_guide**()

Open User Guide. If there is no user guide locally available, open the user guide in Github.

**save_schedule**()

Slot for saving schedule as .schedule

**select_schedule_type**()

Slot for updating schedule view if a schedule type is selected.

**show_about**()

Show the *about* message.

scheduleMain.**main**()

The main method to be runned.

scheduleMain.**my_exception_hook**(*exctype: Type[BaseException]*, *value: BaseException*, *traceback: types.TracebackType*)

Custom exception hook.

**Parameters**

- **exctype** (*type of exception*) – Type of exception catch.
- **value** (*BaseException*) – Exception catched.
- **traceback** (*TracebackType*) – Traceback associated with the exception.

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX