Gramophone User Guide

April 23, 2018

FEMTONICS

ADVANCED MICROSCOPY

Contents

1 Introduction

The Gramophone system is a single dimension locomotion tracking device for head restrained mice that has two modes of operation. It can either be used for hight accuracy velocity recording in conjunction with a two-photon microscope, or as a control interface for behavioural training in a virtual linear maze.

2 Compatibility

The system is compatible with all versions of Microsoft Windows. To run the computer software a Python 3.6 interpreter is required.
compatibility with Femtonics microscopes?

3 Warranty

TODO

4 Contact

If you need further assistance you can contact Femtonics Ltd. via e-mail at info@femtonics.eu or visit our website http://femtonics.eu/contact

5 Cleaning instructions

Disk

Remove the disk for cleaning with the included hex key and clean it with warm soapy water. The disk can be sterilized with alcohol based disinfectants and is resistant to temperatures up to add temp resisitance when available °C.

System

Disconnect the device from the computer by unplugging the USB cable. Clean the device with a damp cloth.

6 Hardware overview

Add full page vector graphic with parts reference

Specifications

TODO

- Operating temperature
- Relative humidity during operation
- Power requirements - 5V DC via USB

Inputs and Outputs

The Gramophone connects to the computer via USB 2.0. A cable is provided with the system but any USB cable with full sized USB-A male and USB-B male connectors can be used.
The system has 5 full sized BNC connectors:

- 4 digital outputs that are connected to ground if their state is set to low, and connected to +5 volts if their state is set to high.
- 2 digital inputs that are considered to be in a low state if the voltage on them is below 2 volts and high if the voltage is above 3 volts. CHECK with electronics department
- 1 analogue output that is configured to give a voltage between 1 and 5 volts that is proportional to the current velocity of the animal. Only available in new firmware

All digital IO is galvanically isolated from the control circuit. Is this clear?

Ground connection

The system has a single banana plug connection on the base plate marked with a ground sign. Always connect this to a ground with the included cable before using the device.

Height adjustment

The distance between the head-holder and the disk can be adjusted to fit mice of any size comfortably.

1. Loosen the lock screws with the included hex key refer to figure
2. Adjust both height adjustment knobs at the same time until the desired height is reached refer to fig. Turning the knobs clockwise lowers the head-holder relative to the disk, while turning it counter-clockwise raises it. chceck with mechanics
3. Lock the position by tightening the lock screws with the included hex key. refer to fig

Position adjustment

The position of the head-holder can be adjusted both sideways and back and forth to position the mouse anywhere on the disk.

Sideways adjustment

1. Loosen the two locking screws on the cylindrical axle with the knobs. refer to fig
2. Move the left part of the head holder to the desired position by sliding the whole assemble on the axle.

3. Lock the left part of the head-holder with the knobs in the desired position.
4. Loosen the two screws (refer to fig) on the right part of the head-holder.
5. Adjust the right part until the head-holder fits a head-mount perfectly. *Note: It is best practice to keep a head-mount handy for this.*
6. Lock the right part in place with the two locking screws refer to fig.

Back and forth adjustment

1. Make sure that the height adjustment screws are tightened. (refer to fig)
2. Remove the screws holding the towers in place (refer to fig) with the included hex key.
3. Position the towers in the desired location such that their centres line up with a pair of mounting holes on the base plate.
4. Put back the removed screws in the new location and tighten them with the included hex key.

7 Software installation

Drivers

No additional drivers are required for the Gramophone system. It operates as a HID (Human Interface Device) for which all drivers are provided by modern operating systems.

Python interpreter

To run the computer software of the Gramophone system a Python 3.6 interpreter has to be installed. To install it download the latest Miniconda installer for your version of Windows (links: **32-bit** or **64-bit**) from the **Conda website**.

Note: If you are not sure which version of Windows you are using you can check by pressing the Win+Pause/Break key combination to open the System window and check under System → System type, or by right clicking the Start menu, choosing System and checking under Device specifications → System type.

Follow the installation steps:

1. Start the downloaded .exe file (Figure 1). Click Next.

Figure 1: The Miniconda installer's welcome screen.

2. Agree to the License Agreement (Figure 2).

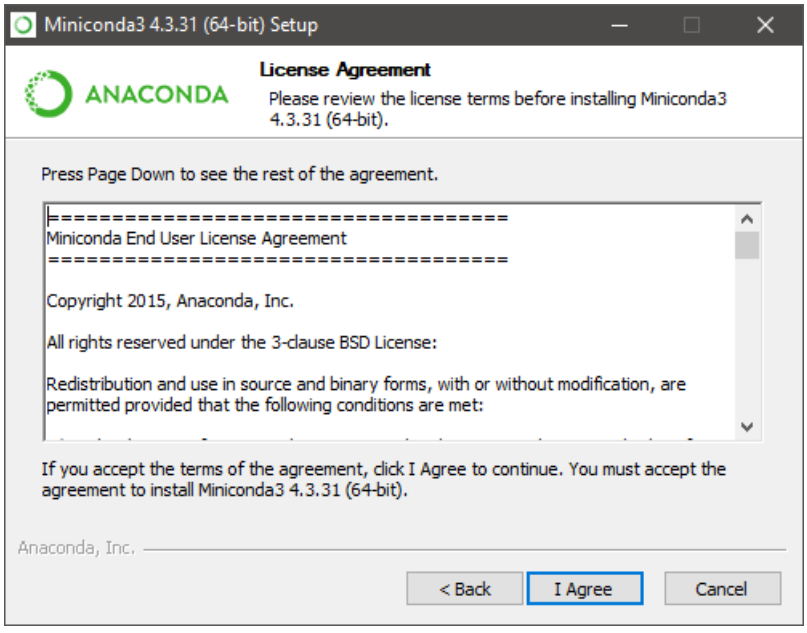

Figure 2: Agree to the License Agreement

3. If multiple users will be using the same machine, install for all users (Figure 3). Click Next.

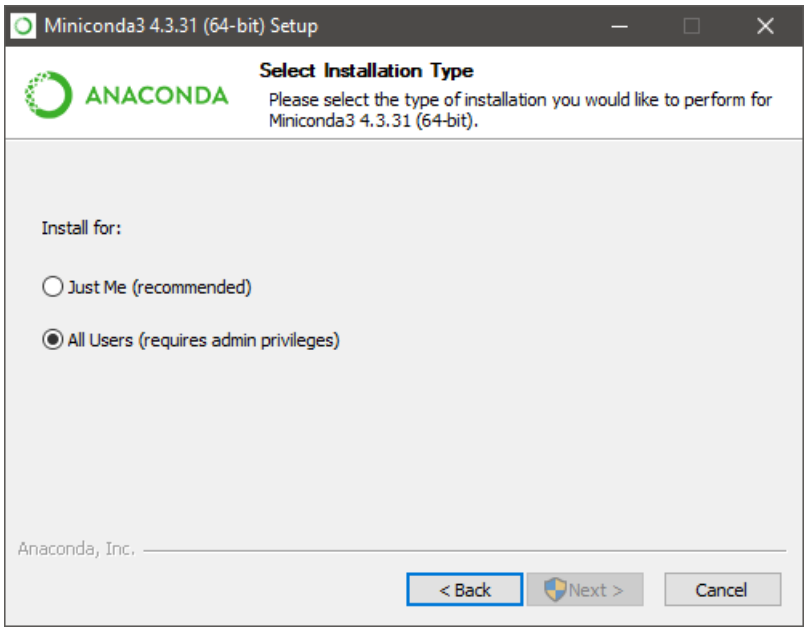

Figure 3: Install for all users if you use multiple users on the computer.

4. Leave the installation directory as the default (Figure 4). Click Next.

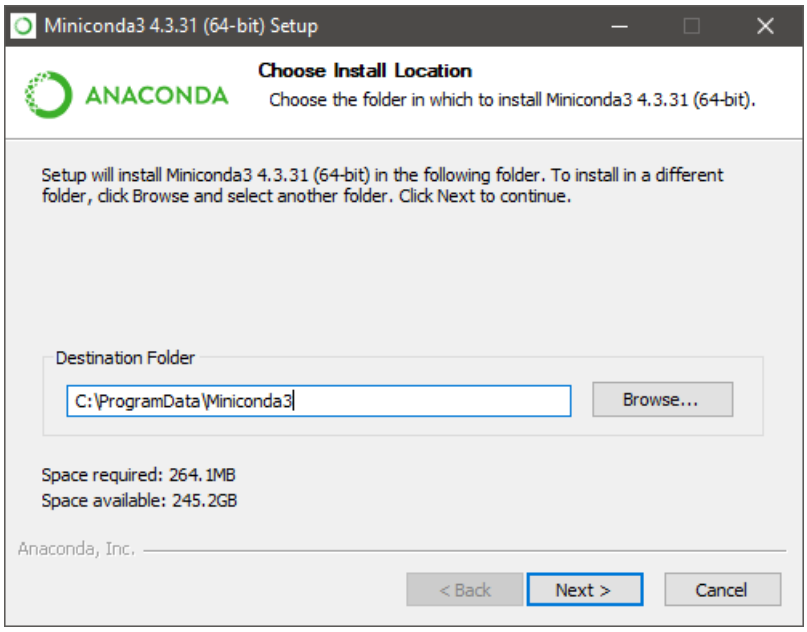

Figure 4: Leave the install directory as the default if possible.

5. Add Anaconda to the system PATH variable and register it as the system's Python 3.2 interpreter (Figure 5). Click Install.

   Note: If you already have other versions of Python installed, leave these options unchecked and make sure you execute Gramophone software from the Anaconda Prompt.

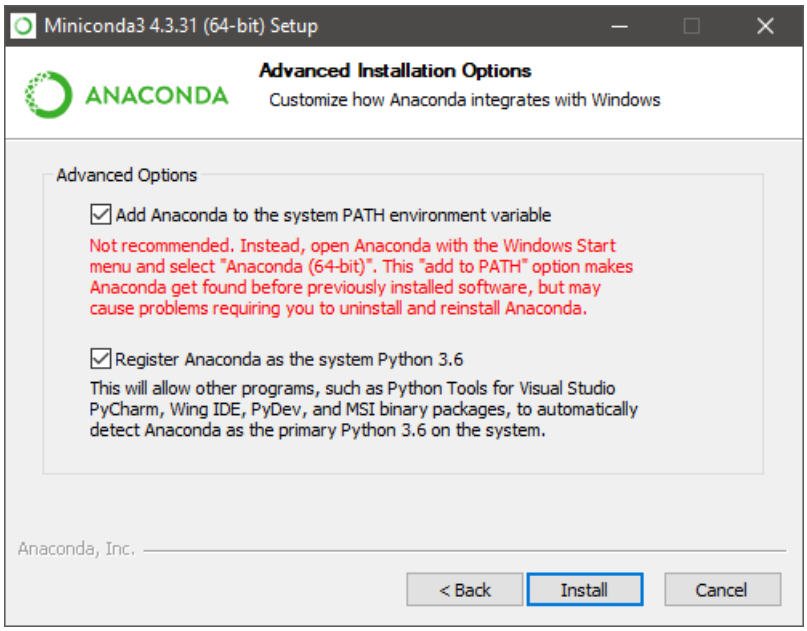

Figure 5: Check both options.

6. Wait for the installation to finish (Figure 6). Click Next.

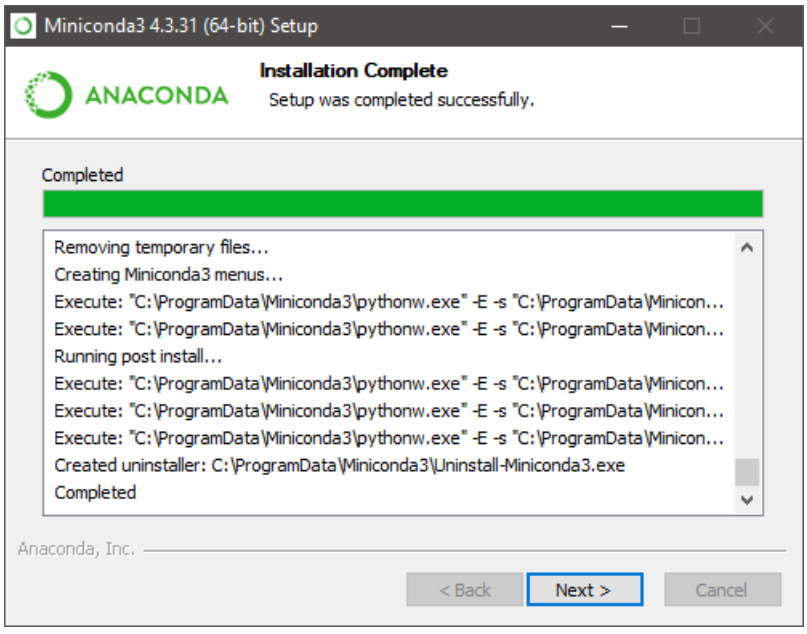

Figure 6: Wait for installation to finish.

7. Exit the installer by clicking Finish (Figure 7).

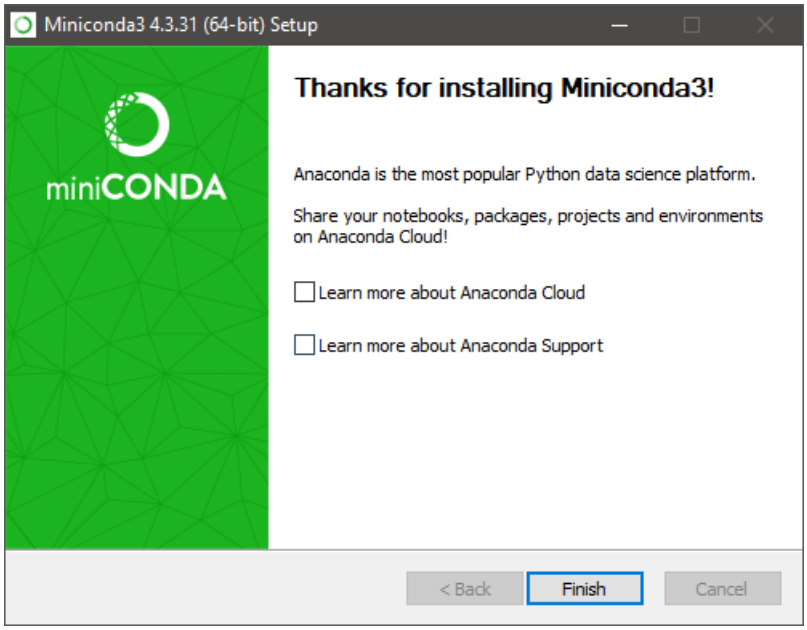

Figure 7: Exit the installer.

Python packages

To install the required packages run the following command in Windows Power Shell, Command Prompt or Anaconda Prompt:

```
pip install dill numpy pyqt5 pyqtgraph pillow pyglet pyserial h5py
```

this will be simplified to: pip install GramophoneTools, once we have the software in the package manager

Associating .py files with the Python interpreter

To start Python scripts (files with .py extension) with a double click in Windows you have to associate them with the Python interpreter (python.exe). To do so right click on any file with a .py extension under the "Open with" menu select the "Choose an other app" option. In the popup window choose the "Look for an other app on this PC" (you might need to open the "More apps" section). In the browser window that opened navigate to the location of python.exe. If you installed Miniconda with the default location this sould be: `C:\ProgramData\Miniconda3\python.exe`

Note: The ProgramData folder is hidden in Windows so you have to type the location in the address bar to navigate there, or make hidden files and folders visible.

Setting environment variables

won't be needed after package management installer is done

To be able to store your level files anywhere you have to add the install location of the LinMaze software to your environment variables.

Open the Start menu and start typing "environment" in the results select the "Edit the system

environment variables". In the System properties window click the "Environment Variables..." button. In the Environment Variables window add a new variable with the name `PYTHONPATH` a the folder of the Python scripts as the value (eg. `C:\VR\LinMaze2`).

8 Gramophone recorder

The Gramophone recorder is software that can make high accuracy velocity recordings. To keep the velocity records in sync with two-photon measurements the digital input 1 is used as a trigger input for recordings.

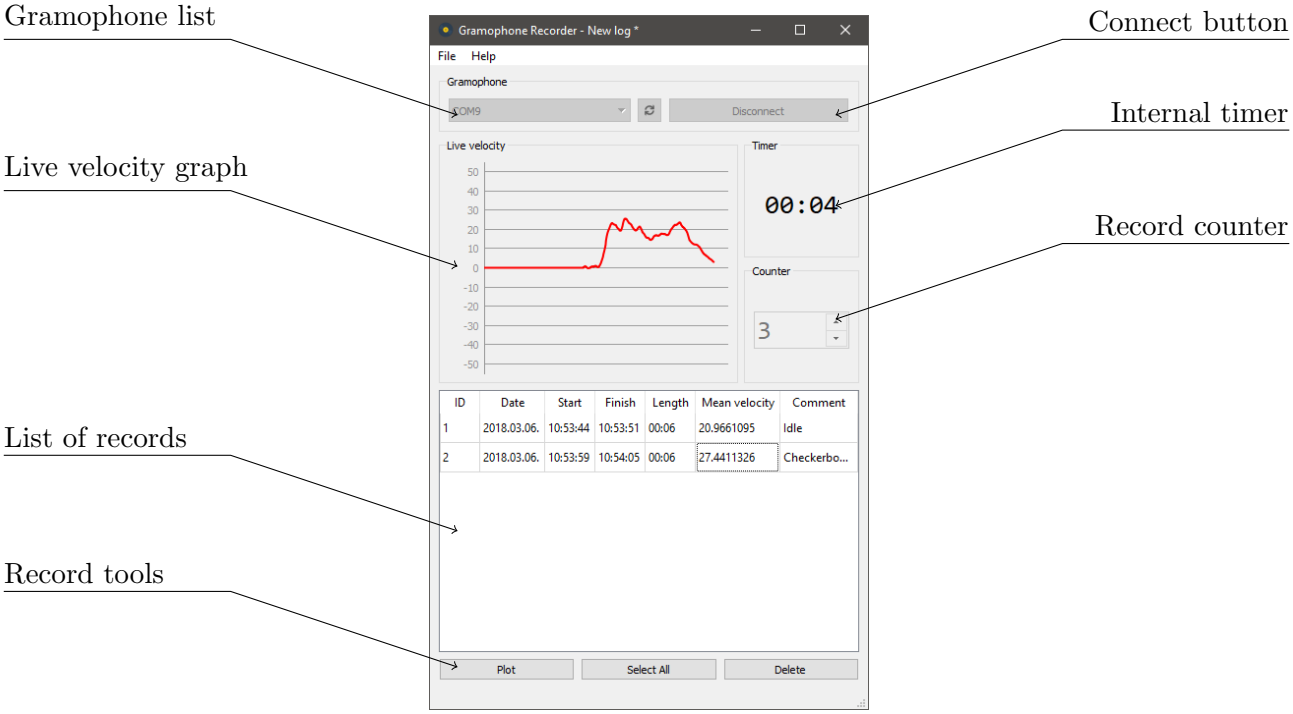

Figure 8: The user interface of the Gramophone recorder during recoring.

Useage

To accurately record velocity with the Gramophone system follow these steps.

Hardware setup

1. Fix the mouse in the head holder. If necessary, adjust the height of the holder as explained in the Height adjustment section.
2. Put the Gramophone with the mouse into the microscope, such that the craniotomy window on the head of the mouse lines up with the microscope's objective perfectly.
3. Connect the grounding point to a ground with the included cable.
4. Connect the digital output of the microscope with digital input #1 reference hardware figure with a BNC cable.

5. Connect the Gramophone to the computer with the included USB cable.

Microscope configuration

Expalin how to configure a trigger in MES/MESc

Software setup

1. Start the Gramophone recorder with the icon on the Desktop or from the terminal with the `gramrecord` command.
2. Select the Gramophone you wish to record from the dropdown list (see: Figure 8). If you can't find the connected device in the list, you can refresh it with the button next to it.
3. Click the Connect button (see: Figure 8). The current velocity will be plotted on the Live velocity plot.
4. Start your measurement protocol in MES or MESc, the recorder will automatically capture velocity for every measurement unit.
5. You can see the velocity records on the List of records (Figure 8). You can select any number of records with the mouse and plot or delete them with the Record tools below the list.

9 LinMaze

LinMaze is a simple virtual linear maze system. It can generate patterns based on parameters and stitch these patterns together into an endless loop the mouse can navigate in using the Gramophone. Optionally you can define rules to condition the mouse to a task eg. to differentiate between different patterns and behave accordingly.

Levels

To use the software you have to make a Python script that defines your Level. A Level is a collection of patterns, events and rules.
To make a new level you have to make a small Pyton script. All levels should start by importing the Level object and an "if" statement that is needed for multithreading to work properly. After the "if" statement you can start constructing your Level. Be sure to put 4 spaces in front of every line from now on.
Note: You can comment out lines by adding a # character in front or make multiple line comment by wrapping with triple ' characters `''' like this '''`. Comments will be coloured green in the following examples.

First create a Level object. In these examples I will call mine "LVL".

```
''' An example level for LinMaze '''
from Level import Level
if __name__ == '__main__':
   LVL = Level(
      name = "myFirstLevel",
      screen_res = (1280, 1024),
      zone_offset = 720,
      transition_width = 100,
      rgb = (0,0.5,0.8)
      )
```

The parameters for the Level are:

- `name` - can be any string, this will show up in default file names
- `screen_res` - two integer numbers in parentheses, separated by commas. These give the resolution of the screen that the level will be running on.
- `zone_offset` - an integer number that gives where the borders of the zones should be on the screen. Eg. If you set it to half the horizontal resolution, zones will be entered when they reach the middle of the screen from the right and are left when the last pixel of the frame leaves the middle.
- `transition_width` - an even integer that describes how wide the smooth transition should be between the frames. Setting this to 100 for example would make all frames 100 pixels bigger on both ends and in this 100 pixel wide space frames would gradually turn into one an other
- `rgb` - 3 float numbers that give the ratio of colours (red, green and blue in order) eg. (1,1,1) is black and white, (0,0.5,0.8) is black and dark cyan.

Frames

Frames are rectangular pictures that can be displayed by the system. There are 8 different types of frames you can add to your level. Frames are always as high as the screen, and as long as the given length parameter.

- ▷ `length` - An integer number that gives the length of the frame in pixels
- ▷ `random_seed` - For randomized frames (eg. clouds and noise). Can be any number. If the random seed of two frames of similar type and length is the same, they will be exact copies. This is useful if you want similar looking patterns. It is an optional parameter, but it is advised to always set it so frames that were generated once can be reused later, significantly reducing loading times.
- ▷ `side_length` - Only for the checkerboard type. An integer number that gives the side length of the squares that make up the checkerboard pattern. If the dimensions of the checkerboard frame are not divisibleby the dimensions of the squares the frame will be cropped (the bottom and right will be cut off) to fit the dimensions
- ▷ `wavelength` - The wavelength (in pixels) of the wave that modulates the grating pattern (ie. the distance between the middle of the white or black lines).
- ▷ `angle` - An angle of the wave's travel in degrees. 0 will result in vertical lines that are rotated in the positive direction (clockwise) as you increase the number (eg you can see 45° lines in the example above)
- ▷ `zone_type` - An optional string that can be used to define what happens when the animal sees this frame. If you leave it blank it defaults to "generic"

Example:

```
# FRAMES
LVL.add_block('binarynoise', length=1000, random_seed=73, zone_type='noise')
LVL.add_block('wood', length=1000, random_seed=73, zone_type='neutral')
LVL.add_block('marble', length=1000, random_seed=73, zone_type='neutral')
LVL.add_block('greynoise', length=1000, random_seed=73, zone_type='noise')
LVL.add_block('square', length=1000, wavelength=200, angle=45, zone_type='right')
LVL.add_block('cloud', length=1000, random_seed=24, zone_type='neutral')
LVL.add_block('checkerboard', length=1000, side_length=45, zone_type='generic')
LVL.add_block('cloud', length=1000, random_seed=25, zone_type='neutral')
LVL.add_block('square', length=1000, wavelength=200, angle=0, zone_type='aversive')
LVL.add_block('cloud', length=1000, random_seed=26, zone_type='neutral')
LVL.add_block('square', length=1000, wavelength=200, angle=45, zone_type='right')
LVL.add_block('square', length=1000, wavelength=200, angle=315, zone_type='left')
LVL.add_block('cloud', length=1000, random_seed=27, zone_type='neutral')
```

Events

Events are things that can happen to the animal. The following table shows the different kinds you can define.

Name	Description	Parameters
`teleport`	Teleport to the given coordinate	`target_position`
`random_teleport`	Teleports to the middle of a random zone with given type (excluding the current one)	`list_of_target_zones`
`port_on`	Pulls the given port high (eg. Opens a valve)	`port`
`port_off`	Pulls the given port low (eg. Closes a valve)	`port`
`start_burst`	Starts a burst pattern according to a given pattern on a given port	`port, on_time, pause_time`
`stop_burst`	Stops the bursting on a given port	`port`
`pause`	Pauses the simulation at a given position. While the simulation is paused, zone rules (seel next chapter) are inactive. Set position to None if you want to pause right where the animal is	`pause_position`
`unpause`	Continues the simulation at a given position. Set position to None if you want to unpause right where the animal is	`unpause_position`

- ▷ `target_position, pause_position, unpause_position` - Integer numbers that gives a target position in pixels.
- ▷ `list_of_target_zones` - A list of strings with all the zone types that are valid targets for this teleport event (eg.: `['neutral', 'left', 'right']`)
- ▷ `port` - The output port used for this event. Can be `'A'`, `'B'` or `'C'`.
- ▷ `on_time` - A parameter for bursting. The time for which the port will be pulled high in seconds.
- ▷ `pause_time` - A parameter for bursting. The time for which the port will be pulled high in seconds.

Example:

```
# EVENTS
LVL.add_event('tp', 'random_teleport', ['right', 'left'])
LVL.add_event('start_puff', 'start_burst', 'B', 0.1, 0.05)
LVL.add_event('stop_puff', 'stop_burst', 'B')
LVL.add_event('pp', 'pause', 0)
LVL.add_event('up', 'unpause', None)
LVL.add_event('open_A', 'port_on', 'A')
LVL.add_event('close_A', 'port_off', 'A')
```

Rules

Rules are the third and last core component of Levels, these are what connect the Events to conditions so that you can define how you want the conditioning to happen. There are 3 different types you can use.

Rule type	Description	Parameters
`zone`	If the animal stays in the given zone for a given time the given event will be triggered.	`zone_type, delay`
`velocity`	If the velocity of the animal is above or below a certain threshold, the given event will trigger.	`vel_rule_type, threshold, delay`
`speed`	If the absolute sum of the last x velocities are above or below a certain threshold, the given event will trigger.	`speed_rule_type, threshold, bin_size`

To add a rule you can call your Level's `add_rule` command like so: `LVL.add_rule(rule_type, event, rule_parameters)`

Example:

```
1  # RULES
2  LVL.add_rule('zone', 'tp', 'neutral', 5)
3  LVL.add_rule('zone', 'start_puff', 'aversive', 3)
4  LVL.add_rule('zone', 'stop_puff', 'neutral', 0)
5  LVL.add_rule('speed', 'pp', 'above', 1000, 100)
6  LVL.add_rule('speed', 'up', 'below', 1000, 100)
7  LVL.add_rule('velocity', 'open_A', 'above', 20, 3)
8  LVL.add_rule('velocity', 'close_A', 'below', 15, 3)
```

The rules defined in this example will result in the following:

- If the mouse spends 5 or more seconds in one of the neutral zones it will be teleported out to a left or right zone.
- If the mouse spends 3 or more seconds in the aversive zone the valve connected to port B will start bursting as defined in the `start_puff` event
- If the mouse enters a neutral zone the bursts of port B will stop immediately
- The simulation will pause if the absolute sum of the last 100 velocities is above 1000 and unpause again if it gets below 1000
- If the mouse runs with a higher velocity than 20 for at least 3 seconds, port A will be opened.
- If the mouse slows down and stays below 15 velocity, for at least 3 seconds port A will close

Saving a preview image

When you finished making your level you can save a 1:1 ratio image of it with the `Level.save_image()` command (eg. `LVL.save_image()`). When the program is run you will be given a file selection window to give the location and filename of the image.

Saving a summary

You can save a human readable summary text with the `Level.save_summary()` command of your level. You will be presented with a file selection window to specify the location and filename of the summary file (eg. `LVL.save_summary()`).

Playing the Level

If you have a Level made you can call it's `play()` command to start the simulation. By default this would start the simulation with the primary monitor as the left screen, with the first gramophone as an input, with a gramophone speed to VR speed ratio of 1:1, with black and white colours on a single monitor and it would run until you press escape or close the simulation window. You can change the defaults by setting the corresponding, optional argument of the play command to something else.
The optional arguments are:

- `vel_ratio` - float number. The velocity read from the gramophone gets multiplied by this. Setting it to a negative value changes the forward direction.
- `runtime_limit` - float number. How long should the simulation run (in minutes).
- `left_monitor` - integer number or None. Which monitor is used as a left side display.
- `right_monitor` - integer number or None. Which monitor is used as a right side display.
- `gramofon_port` - the COM port the gramophone is connected to as a string eg. 'COM3' or None for automatic selection of the first connected gramophone. final product will have HID devices not COM ports
- `fullscreen` - True for full screen and False for windowed mode.

Example:

```
LVL.play(left_monitor=None, right_monitor=1, vel_ratio=2, gramofon_port='COM3',
    runtime_limit=5, fullscreen=True)
```

Simulation results

When you call the play command you will be presented with a file selection window, on which you can select the location and filename of your log file. The log is automatically saved every 10 seconds or on the end of the simulation (on proper exit, do not close the console window, close the simulation window or select it and exit with the Esc key).

the HDF5 structure will change as the final product has more IO. also, move this to the dev docs instead
The log file is saved with a .vrl extension. It is a HDF5 file with the following structure:

- `analogue_input` - the state of the analogue input [0,1024]
- `g_time` - the time of the microcontroller
- `paused` array of zeros and ones, value is 1 if the simulation was paused at that point
- `ports` - the satte of the 3 outputs in separate arrays, 1 if the output was high, 0 if it was low

  - `A` - the states of port A
  - `B` - the states of port B
  - `C` - the states of port C
- `position` - the position in the maze in pixels
- `teleport` - array of zeros and ones, 1 if there was a teleport at that point
- `time` - time axis, array float values of the computers time
- `velocity` - array of signed integers with the velocity in pixels/record
- `zone` - n×m matrix of ones and zeros. Each column is an array of ones and zeros for that zone
- `zone_types` - a group of arrays of zeros and ones for each zone type that was defined
  - `zone_type_1` - 1 when the mouse was in a type_1 zone
  - `zone_type_2` - 1 when the mouse was in a type_2 zone
  - `Etc.`

10 Release notes

too technical, move to dev docs?

LinMaze changelog

2018.03.01.

- The play command no longer has `screen_number` and `dual_screen` arguments. Instead it has a `left_screen` and `right_screen` arguments. `left_monitor` is the number of the monitor that is on the left side of the mouse, `right_monitor` is the number of the one on the right (its the mirror of the left). These can be a number or None if there is only a monitor on one side. `right_monitor` is None by default. If you want everything to be the same change `screen_number` to `left_monitor` in your levels. Sorry for the inconvenience!
- Logging is changed. Instead of a .csv file you get a .vrl file. This is a structured HDF5 that stores more data about your session. The log is automatically saved every 10 seconds but closing the console is still not advisable (exit with escape or close the simulation window on the task-bar)
- The play command now has a `fullscreen` option. It is True by default, if you set it to False the simulation will run in a window (handy for testing your level).
- The LinMaze related modules in MES RozsaLab toolbox now work with .vrl files too.
- Updated the user guide with these changes. I have also attached it. The new things are in the "Playing your level" and "Simulation results" chapters.
- The initial filename of the log file now includes the name of the level in parentheses

2017.12.12.

- Random teleports can no longer land in the same zone type as the starting zone (eg. no teleport from cloud to cloud, left to left etc.). Such events caused the animal to be stuck in a zone after teleport, because the rule didn't reset.
- The save window now initially opens in the same location you run your level from (instead of `C:\VR\...`).

2017.09.28. (LinMaze 2.0)

- Levels are no longer needed to be saved, you can just run the .py file that describes it.
- The rendering of levels is now done with multiprocessing, making it marginally faster
- The frames made are now locally cached on the computer, so the software never does the same calculation twice
- A new parameter (`transition_width`) was added to control the width of the transition area, so sharp transitions can be avoided (the example attached has 100 pixel wide transitions)
- A new parameter (rgb) was added to control the proportion of the displayed colors. eg. setting it to (0,0,1) has the same effect as the currently used blue foils in front of the screens.

- During the operation of the VR only relevant messages will be shown. eg. if a port is already closed, the close port event won't trigger again
- All save operations now display an operating system specific save window. This window also handles overwrite warnings and suggests names for your files.
- The data received from the Gramophone is now filtered, resulting in a smoother velocity curve. This is also visible during the VR operation, the displayed image is less choppy.
- The state of the A1 analogue input of the Arduino is now recorded during the simulation. You can attach a lick port or other sensor here and its value will be saved.
- The internal clock of the Arduino is reset on the start of the simulation and its value is also recorded in the log.
- You can now save a human readable summary of your level with the `Level.save_summary()` command.