# Installing Dependencies

Required:

- Python 3.x
- Python Libraries:
    - NumPy - https://numpy.org/install/
    - SciPy - https://scipy.org/install/
    - NetworkX - https://networkx.org/documentation/stable/install.html
    - nltk - https://www.nltk.org/install.html

Optional:

- Matlab (for simulation purposes)
- Unity (for simulation purposes)
- Python Libraries
    - Spot - https://spot.lrde.epita.fr/install.html

# Overview

1. Write an Event-based STL specification in the correct format
2. Prepare the specification for execution
3. Manually generate a Büchi automaton if Spot is not installed
4. Simulate an execution of a specification

# Writing Specifications

- Syntax of an Event-based STL formula $\Psi$

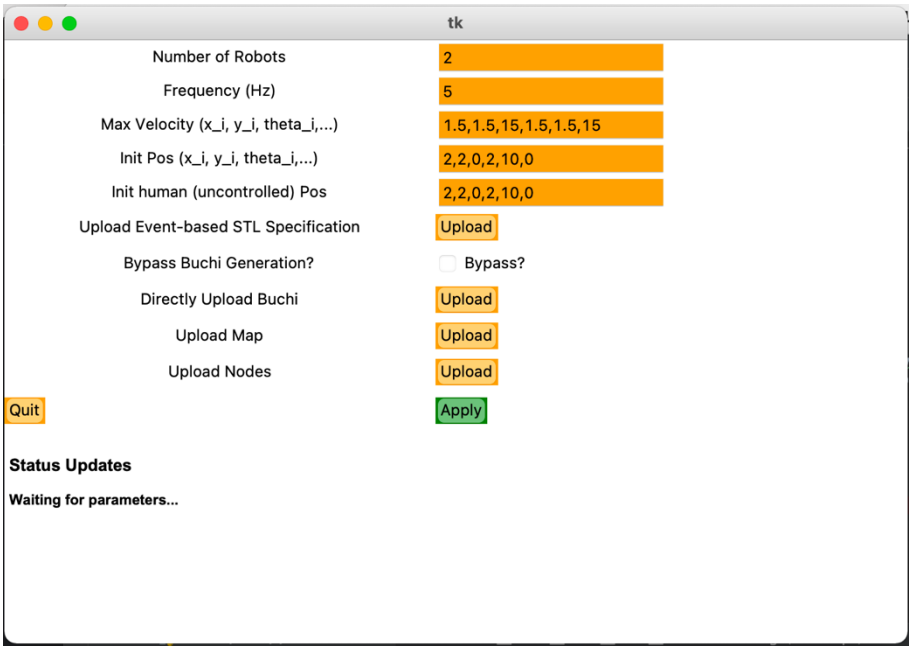
$$
\begin{aligned}
\varphi &::= \mu \mid \neg\mu \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\
\alpha &::= \pi \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2 \\
\Psi &::= G_{[a,b]}\ \varphi \mid F_{[a,b]}\ \varphi \mid \varphi_1\ U_{[a,b]}\ \varphi_2 \mid \\
&\quad\ G(\alpha \Rightarrow\ \Psi) \mid \Psi_1 \wedge \Psi_2 \mid \Psi_1 \vee \Psi_2
\end{aligned}
$$

- Temporal operators are written as alw_[a,b], ev_[a,b], or un_[a,b]
- All robots are assumed to have states x, y, theta represented as pos[0], pos[1], pos[2]
- Parentheses are only used for grouping of predicates, temporal operators, or events
- Predicates should not contain any spaces (ex. “sqrt[[pos[0]-2]^2+[pos[1]-2]^2]<1” )
- Temporal operators and timing bounds should not contain spaces (ex. ev_[0,10])
- All predicates grouped with a temporal operator are surrounded by parenthesis (ex. “(ev_[5,10] pos[0]>1 | alw_[0,5] pos[1]<2)” )
- All environment events are grouped with parentheses (ex. “((alarm1 & alarm2) | (alarm3 | alarm4))
- The specification is save as a 1 line .txt file

- Example 1:
    - G(alarm1 => (ev_[0,10] (pos[0]>10 | pos[3]>11)))
- Example 2:
    - G(((alarm1 & alarm2) => ((ev_[0,10] sqrt[[pos[0]-17]^2+[pos[1]-2]^2]<1) | (ev_[0,10] sqrt[[pos[3]-17]^2+[pos[4]-2]^2]<1))))

## Preparing a Specification

To prepare a specification for execution, run the file “runEvBasedSTL.py”. This script will open a GUI and ask for several different inputs.




- **Number of Robots**: Number of controllable robots that the specification includes
- **Frequency (Hz)**: Frequency at which the specification is run.
- **Max Velocity**: Velocity bounds for each robot (x,y,theta)
- **Initial Pos**: Initial position of each controllable robot (for simulation purposes)
- **Init human**: Initial position of dynamic obstacles (humans, uncontrolled robots, etc.)
- **Upload Event-based STL Specification**: Upload a txt file that contains the one line specification. The specification must be in the form described in the “Writing Specifications” section of this document
- **Bypass Büchi Generation**?: By default, the script will use Spot to generate a Büchi automaton from the given specification. If Spot is not installed, the Büchi can be uploaded as a separate txt file. The Büchi can be obtained from the online tool on the Spot website.
- **Directly Upload Büchi:** If the bypass option is checked, upload the txt file containing the Büchi automaton generated by the online tool on the Spot website.
- **Upload map**: upload an environment map. The map should be a .txt file with each line representing a line segment [x1 y1 x2 y2]

    - **Upload nodes**: upload a roadmap for the map. The nodes should be in a .txt file with each line representing a point [x y]

- **Options within runEvBasedSTL**:
    - Variable *loadOnStart*: If set to “0” the GUI will open to create a new pickle file. If set to “1” the pickle file will be loaded to be used for message debugging (More info in the “Executing a Specification” section of this document).
    - viewTree: If set to 1 the parse tree will be printed. This Is useful to ensure the tree is correct and the specification is in the correct format
- **Output**
    - The output of this script is a pickle file and mat file that contains the information necessary to execute a specification. The pickle file is saved in the “pickle files” folder and the mat file is saved in the “matlab files” foleder.
    - The name of the files is the name of the .txt file for the specification

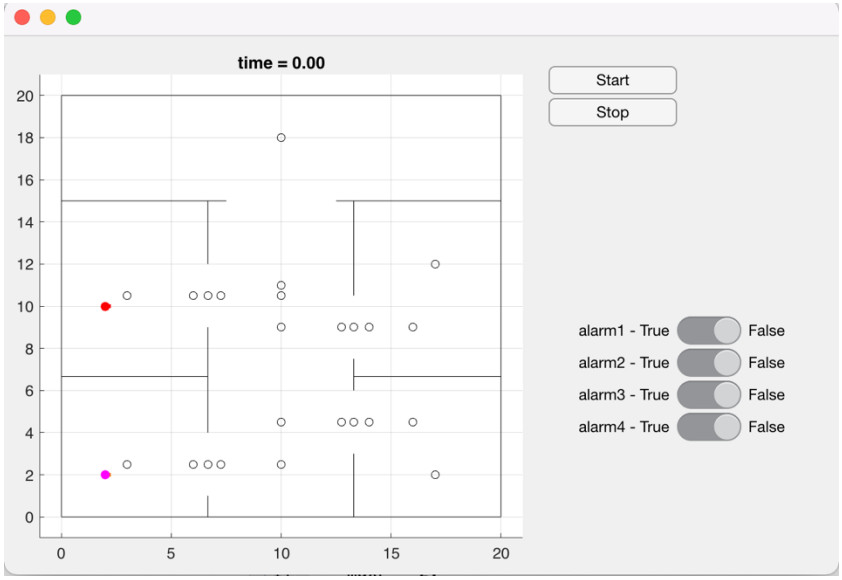
## Generating a Büchi Automaton using Spot's online tool

- Select the Bypass option on the GUI after running runEvBasedSTL.py
- After pressing "Apply" the abstracted Event-based STL specification will be printed
- Visit https://spot.lrde.epita.fr/app/ and enter the printed specification
- Select "Acceptance: (State-based) Büchi " and "complete"
- After pressing enter to generate the automaton, select "NEVERCLAIM" and copy the output into a txt file.
- To generate the pickle file, run "runEvBasedSTL.py", select "bypass", and upload the txt file with the Büchi generated from Spot.

## Executing a specification

- A specification is executed through simulation in Unity (clientUnity.py) or Matlab (clientMatlab.py).
- **Running in Matlab**
    - In the file “clientMatlab.py” change the *pickle_file_path* variable to the pickle file for your specification
    - In a terminal run “clientMatlab.py”. The client will attempt to connect to Matlab to begin the simulation
    - In Matlab run the script “RunTCPSim.m”. The mat file that is loaded should match pickle file

- A simulation window will appear with a start button to begin the simulation and input buttons which represent environment events.




- **Running in Unity**
  - In the file “clientUnity.py” change the *pickle_file_path* variable to the pickle file for your specification
  - In a terminal run “clientUnity.py”. The client will attempt to connect to Unity to begin the simulation
  - To run a simulation in Unity attach the file “TCPServer.cs” to a robot in a Unity environment.
  - When you begin the game in Unity, the server will connect and the execution will begin
- **Debugging a Specification**
  - At each time step a message is printed from the Unity or Matlab console. This string contains information about the state of the robots and the system that are used to generate control. To debug a specification change the *debugMessage* variable to “1” in “runEvBasedSTL.py” and change the *Mes* variable to the string that was printed from the Matlab/Unity console. This will allow you to run the specification at the time step where an error occurred.