²JST-CREST / IEEE-RAS Spring School on "Social and Artificial Intelligence for User-Friendly Robots"

## Programming NAO to Learn Behaviors

NAOqi Doc
We can find the documentation for the version of our robot in the next link:

http://doc.aldebaran.com/

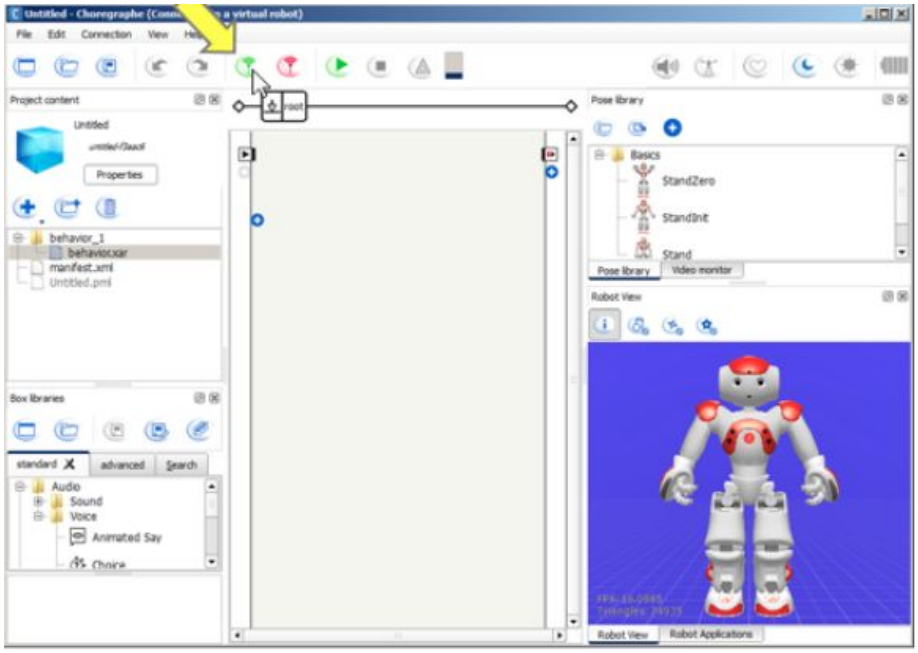
# Getting Started

**Choregraphe** is a multi-platform desktop application, allowing you to:

- Create animations, behaviors and dialogs,
- Test them on a simulated robot, or directly on a real one,
- Monitor and control you robot,
- Enrich Choregraphe behaviors with your own Python code.

Hello World 1 - using Choregraphe

-




# Python SDK - Overview

The Python API for **Softbank Robotics** robots allows you to:

- use all of the C++ API from a remote machine, or
- create Python modules that can run remotely or on the robot.Using Python is one of the easiest ways to program with **Softbank Robotics** robots.

NAOqi API 2.5
http://doc.aldebaran.com/2-5/index_dev_guide.html

We are going to focus in the Submodules in **bold**.

Task:
Participants will have to complete some part of the code to make use of some of the modules.

- NAOqi Core
  - ALBehaviorManager API | overview
  - ALConnectionManager API | overview
  - ALDiagnosis API | overview
  - ALExpressionWatcher API | overview
  - ALExtractor API | overview
  - ALKnowledge API | overview
  - **ALMemory** API | overview
  - ALModule API | overview

- ALMood API | overview
- ALNotificationManager API | overview
- ALPreferenceManager API | overview
- ALResourceManager API | overview
- ALSystem API | overview
- ALVisionExtractors API | overview
- ALTabletService API | overview
- ALUserInfo API | overview
- ALUserSession API | overview
- ALWorldRepresentation API | overview
- PackageManager API | overview
- ServiceManager API

**ALMemory**

```python
#! /usr/bin/env python
# -*- encoding: UTF-8 -*-

"""Example: A Simple class to get & read FaceDetected Events"""

import qi
import time
import sys
import argparse


class HumanGreeter(object):
    """
    A simple class to react to face detection events.
    """

    def __init__(self, app):
        """
        Initialisation of qi framework and event detection.
        """
        super(HumanGreeter, self).__init__()
        app.start()
        session = app.session
        # Get the service ALMemory.
        self.memory = session.service("ALMemory")
        # Connect the event callback.
        self.subscriber = self.memory.subscriber("FaceDetected")
        self.subscriber.signal.connect(self.on_human_tracked)
        # Get the services ALTextToSpeech and ALFaceDetection.
        self.tts = session.service("ALTextToSpeech")
        self.face_detection = session.service("ALFaceDetection")
        self.face_detection.subscribe("HumanGreeter")
        self.got_face = False

    def on_human_tracked(self, value):
        """
        Callback for event FaceDetected.
        """
        if value == []:  # empty value when the face disappears
            self.got_face = False
        elif not self.got_face:  # only speak the first time a face appears
            self.got_face = True
            print "I saw a face!"
            self.tts.say("Hello, you!")
            # First Field = TimeStamp.
            timeStamp = value[0]
            print "TimeStamp is: " + str(timeStamp)

            # Second Field = array of face_Info's.
            faceInfoArray = value[1]
            for j in range( len(faceInfoArray)-1 ):
                faceInfo = faceInfoArray[j]

                # First Field = Shape info.
                faceShapeInfo = faceInfo[0]

                # Second Field = Extra info (empty for now).
                faceExtraInfo = faceInfo[1]

                print "Face Infos :  alpha %.3f - beta %.3f" % (faceShapeInfo[1], faceShapeInfo[2])
                print "Face Infos :  width %.3f - height %.3f" % (faceShapeInfo[3], faceShapeInfo[4])
                print "Face Extra Infos :" + str(faceExtraInfo)

    def run(self):
        """
        Loop on, wait for events until manual interruption.
        """
        print "Starting HumanGreeter"
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print "Interrupted by user, stopping HumanGreeter"
            self.face_detection.unsubscribe("HumanGreeter")
            #stop
            sys.exit(0)
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")

    args = parser.parse_args()
    try:
        # Initialize qi framework.
        connection_url = "tcp://" + args.ip + ":" + str(args.port)
        app = qi.Application(["HumanGreeter", "--qi-url=" + connection_url])
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) +".\n"
               "Please check your script arguments. Run with -h option for help.")
        sys.exit(1)

    human_greeter = HumanGreeter(app)
    human_greeter.run()
```

1. Connect to the robot
    - import qi
    - connection_url = "tcp://" + args.ip + ":" + str(args.port)
    - app = qi.Application(["HumanGreeter", "--qi-url=" + connection_url])
2. Start Application and call session services
    - app.start()
    - session = app.session
    - self.memory = session.service("ALMemory")
3. Connect the event callback.
    - self.subscriber = self.memory.subscriber("FaceDetected")
    - self.subscriber.signal.connect(self.on_human_tracked)

Other services:

- self.tts = session.service("ALTextToSpeech")
- self.tts.setVolume(0.2)
- self.tts.say("Hello, you!")
- self.face_detection = session.service("ALFaceDetection")
- self.face_detection.subscribe("HumanGreeter")

[NAOqi Interaction engines]

- ALAutonomousLife
- ALAutonomousBlinking
- ALBackgroundMovement
- **ALBasicAwareness**
- ALListeningMovement
- ALSpeakingMovement
- ALDialog

## ALAutonomousLife

```python
#! /usr/bin/env python
# -*- encoding: UTF-8 -*-

"""Example: Use setAutonomousAbilityEnabled Method"""

import qi
import argparse
import sys


def main(session):
    """
    This example uses the setAutonomousAbilityEnabled method.
    """
    # Get the service ALAutonomousLife.

    life_service = session.service("ALAutonomousLife")
    life_service.setAutonomousAbilityEnabled("BasicAwareness", True)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")

    args = parser.parse_args()
    session = qi.Session()
    try:
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) +".\n"
               "Please check your script arguments. Run with -h option for help.")
        sys.exit(1)
    main(session)
```

**ALBasicAwarenes**

```
#! /usr/bin/env python
# -*- encoding: UTF-8 -*-

"""Example: Use startAwareness and stopAwareness Methods"""

import qi
import argparse
import sys
import time


def main(session):
    """
    This example uses the setEnabled method.
    """
    # Get the services ALBasicAwareness and ALMotion.

    ba_service = session.service("ALBasicAwareness")
    motion_service = session.service("ALMotion")

    print "Waiting for the robot to be in wake up position"
    motion_service.wakeUp()

    print "Starting BasicAwareness"
    ba_service.setEnabled(True)

    print "Take some time to play with the robot"
    time.sleep(30)

    print "Stopping BasicAwareness"
    ba_service.setEnabled(False)

    print "Waiting for the robot to be in rest position"
    motion_service.rest()


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: \
                        use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")

    args = parser.parse_args()
    session = qi.Session()
    try:
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " +
               str(args.port) + ".\n"
               "Please check your script arguments. Run with -h option for \
help.")
        sys.exit(1)

    main(session)
```

1. Session service
    - self.basic_awareness = session.service("ALBasicAwareness")
    - self.posture = session.service("ALRobotPosture")
    - self.motion = session.service("ALMotion")
2. Signals human tracked and lost
    - subscriber = self.memory.subscriber(event_name)
    - subscriber.signal.connect(callback_func)
3. Speech recognition
    - self.speech_reco.setVocabulary(["yes", "no"], False)
4. Human position
    - memory_key = "PeoplePerception/Person/" + str(id_person_tracked) + \ "/PositionInWorldFrame"

[NAOqi Motion]

- ALAnimationPlayer
- **ALRobotPosture**
- ALNavigation
- ALRecharge
- **ALMotion**
- **ALTracker**
- ALMotionRecorder

```python
#! /usr/bin/env python
# -*- encoding: UTF-8 -*-

"""Example: Use goToPosture Method"""

import qi
import argparse
import sys


def main(session):
    """
    This example uses the goToPosture method.
    """
    # Get the service ALRobotPosture.

    posture_service = session.service("ALRobotPosture")

    posture_service.goToPosture("StandInit", 1.0)
    posture_service.goToPosture("SitRelax", 1.0)
    posture_service.goToPosture("StandZero", 1.0)
    posture_service.goToPosture("LyingBelly", 1.0)
    posture_service.goToPosture("LyingBack", 1.0)
    posture_service.goToPosture("Stand", 1.0)
    posture_service.goToPosture("Crouch", 1.0)
    posture_service.goToPosture("Sit", 1.0)

    print posture_service.getPostureFamily()


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")

    args = parser.parse_args()
    session = qi.Session()
    try:
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) +".\n"
               "Please check your script arguments. Run with -h option for help.")
        sys.exit(1)
    main(session)
```

## ALTracker

```
#! /usr/bin/env python
# -*- encoding: UTF-8 -*-

"""Example: Use Tracking Module to Track a Face"""

import qi
import argparse
import sys
import time


def main(session, faceSize):
    """
    This example shows how to use ALTracker with face.
    """
    # Get the services ALTracker and ALMotion.

    motion_service = session.service("ALMotion")
    tracker_service = session.service("ALTracker")

    # First, wake up.
    motion_service.wakeUp()

    # Add target to track.
    targetName = "Face"
    faceWidth = faceSize
    tracker_service.registerTarget(targetName, faceWidth)

    # Then, start tracker.
    tracker_service.track(targetName)

    print "ALTracker successfully started, now show your face to robot!"
    print "Use Ctrl+c to stop this script."

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        print
        print "Interrupted by user"
        print "Stopping..."

    # Stop tracker.
    tracker_service.stopTracker()
    tracker_service.unregisterAllTargets()
    motion_service.rest()

    print "ALTracker stopped."


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")
    parser.add_argument("--facesize", type=float, default=0.1,
                        help="Face width.")

    args = parser.parse_args()
    session = qi.Session()
    try:
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) +".\n"
               "Please check your script arguments. Run with -h option for help.")
        sys.exit(1)
    main(session, args.facesize)
```

1. Session service
    - tracker_service = session.service("ALTracker")
2. Add target to track and set mode
    - targetName = "Face"
    - faceWidth = faceSize
    - tracker_service.registerTarget(targetName, faceWidth)
3. Stop tracker
    - tracker_service.stopTracker()
    - tracker_service.unregisterAllTargets()

- NAOqi Audio
    - ALAnimatedSpeech
    - ALAudioDevice
    - ALAudioPlayer
    - ALAudioRecorder
    - ALSoundDetection
    - ALSoundLocalization
    - **ALSpeechRecognition**
    - **ALTextToSpeech**
    - ALVoiceEmotionAnalysis
    - ALAudioSourceLocalization

**ALSpeechRecognition**

```python
#! /usr/bin/env python
# -*- encoding: UTF-8 -*-

"""Example: Use ALSpeechRecognition Module"""

import qi
import argparse
import sys
import time


def main(session):
    """
    This example uses the ALSpeechRecognition module.
    """
    # Get the service ALSpeechRecognition.

    asr_service = session.service("ALSpeechRecognition")

    asr_service.setLanguage("English")

    # Example: Adds "yes", "no" and "please" to the vocabulary (without wordspotting)
    vocabulary = ["yes", "no", "please"]
    asr_service.setVocabulary(vocabulary, False)

    # Start the speech recognition engine with user Test_ASR
    asr_service.subscribe("Test_ASR")
    print 'Speech recognition engine started'
    time.sleep(20)
    asr_service.unsubscribe("Test_ASR")


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")

    args = parser.parse_args()
    session = qi.Session()
    try:
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) +".\n"
               "Please check your script arguments. Run with -h option for help.")
        sys.exit(1)
    main(session)
```

## ALTextToSpeech

```python
# Creates a proxy on the text-to-speech module
from naoqi import ALProxy

IP = "<IP ADDRESS>"
tts = ALProxy("ALTextToSpeech", IP, 9559)
```

```python
# Example: Sends a string to the text-to-speech module
tts.say("Hello World!")
```

```python
tts.setParameter("speed", 200)
tts.resetSpeed()
```

```
tts.setParameter("doubleVoice", 1)
tts.setParameter("doubleVoiceLevel", 0.5)
tts.setParameter("doubleVoiceTimeShift", 0.1)
tts.setParameter("pitchShift", 1.1)
```

1. Session services
    - self.memory = session.service("ALMemory")
    - self.tts = session.service("ALTextToSpeech")
    - self.asr_service = session.service("ALSpeechRecognition")
2. Set Language
    - self.asr_service.setLanguage("English")
3. Declare vocabulary
    - vocabulary = ["yes", "no", "hello", "please"]
    - self.asr_service.pause(True)
    - self.asr_service.setVocabulary(vocabulary, False)
    - self.asr_service.pause(False)
4. Signal
    - self.subscriber_speech = self.memory.subscriber("WordRecognizedAndGrammar")
    - self.subscriber_speech.signal.connect(self.on_word_recognized)
    - Word: ['no', 0.44699999690055847, 'modifiable_grammar']

- NAOqi Vision
    - ALBacklightingDetection
    - ALBarcodeReader
    - ALColorBlobDetection
    - **ALDarknessDetection**
    - ALLandMarkDetection
    - ALMovementDetection
    - ALPhotoCapture
    - ALRedBallDetection
    - ALSegmentation3D
    - ALVideoDevice
    - ALVideoRecorder
    - ALVisionRecognition
    - ALLocalization
    - ALVisualCompass
    - ALVisualSpaceHistory
    - ALCloseObjectDetection

```
#include <alproxies/aldarknessdetectionproxy.h>
```

Event: DarknessDetection/DarknessDetected()

Memory Key: DarknessDetection/DarknessValue

- NAOqi People Perception
  - ALEngagementZones
  - ALFaceCharacteristics
  - **ALFaceDetection**
  - ALGazeAnalysis
  - ALPeoplePerception
  - ALSittingPeopleDetection
  - ALWavingDetection

```
# This test demonstrates how to use the ALFaceDetection module.
# Note that you might not have this module depending on your distribution
#
# - We first instantiate a proxy to the ALFaceDetection module
#     Note that this module should be loaded on the robot's NAOqi.
#     The module output its results in ALMemory in a variable
#     called "FaceDetected"
# - We then read this ALMemory value and check whether we get
#   interesting things.
import time
from naoqi import ALProxy

# Replace this with your robot's IP address
IP = "10.0.252.91"
PORT = 9559

# Create a proxy to ALFaceDetection
try:
  faceProxy = ALProxy("ALFaceDetection", IP, PORT)
except Exception, e:
  print "Error when creating face detection proxy:"
  print str(e)
  exit(1)

# Subscribe to the ALFaceDetection proxy
# This means that the module will write in ALMemory with
# the given period below
period = 500
faceProxy.subscribe("Test_Face", period, 0.0 )
```

```
# ALMemory variable where the ALFaceDetection module
# outputs its results.
memValue = "FaceDetected"

# Create a proxy to ALMemory
try:
  memoryProxy = ALProxy("ALMemory", IP, PORT)
  except Exception, e:
  print "Error when creating memory proxy:"
  print str(e)
  exit(1)

# A simple loop that reads the memValue and checks whether faces are detected.
for i in range(0, 20):
  time.sleep(0.5)
  val = memoryProxy.getData(memValue, 0)
  print ""
  print "\*****"
  print ""

# Check whether we got a valid output: a list with two fields.
if(val and isinstance(val, list) and len(val) == 2):
  # We detected faces !
  # For each face, we can read its shape info and ID.
  # First Field = TimeStamp.
  timeStamp = val[0]
  # Second Field = array of face_Info's.
  faceInfoArray = val[1]

  try:
  # Browse the faceInfoArray to get info on each detected face.
    for faceInfo in faceInfoArray:
    # First Field = Shape info.
    faceShapeInfo = faceInfo[0]
    # Second Field = Extra info (empty for now).
    faceExtraInfo = faceInfo[1]
    print "  alpha %.3f - beta %.3f" % (faceShapeInfo[1], faceShapeInfo[2])
    print "  width %.3f - height %.3f" % (faceShapeInfo[3], faceShapeInfo[4])
  except Exception, e:
    print "faces detected, but it seems getData is invalid. ALValue ="
    print val
    print "Error msg %s" % (str(e))
else:
  print "Error with getData. ALValue = %s" % (str(val))
  # Unsubscribe the module.

faceProxy.unsubscribe("Test_Face")
print "Test terminated successfully."
```

- NAOqi Sensors & LEDs
    - ALTactileGesture
    - ALBattery
    - ALBodyTemperature
    - ALChestButton
    - ALFsr
    - **ALTouch**
    - ALLaser
    - ALLeds
    - ALSensors
    - **ALSonar**

## ALTouch

```
# -*- encoding: UTF-8 -*-

"""Example: Say `My {Body_part} is touched` when receiving a touch event"""

import qi
import argparse
import functools
import sys

class ReactToTouch(object):
    """ A simple module able to react
        to touch events.
    """
    def __init__(self, app):
        super(ReactToTouch, self).__init__()

        # Get the services ALMemory, ALTextToSpeech.
        app.start()
        session = app.session
        self.memory_service = session.service("ALMemory")
        self.tts = session.service("ALTextToSpeech")
        # Connect to an Naoqil Event.
        self.touch = self.memory_service.subscriber("TouchChanged")
        self.id = self.touch.signal.connect(functools.partial(self.onTouched, "TouchChanged"))

    def onTouched(self, strVarName, value):
        """ This will be called each time a touch
        is detected.

        """
        # Disconnect to the event when talking,
        # to avoid repetitions
        self.touch.signal.disconnect(self.id)

        touched_bodies = []
        for p in value:
            if p[1]:
                touched_bodies.append(p[0])

        self.say(touched_bodies)

        # Reconnect again to the event
        self.id = self.touch.signal.connect(functools.partial(self.onTouched, "TouchChanged"))

    def say(self, bodies):
        if (bodies == []):
            return

        sentence = "My " + bodies[0]

        for b in bodies[1:]:
            sentence = sentence + " and my " + b

        if (len(bodies) > 1):
            sentence = sentence + " are"
        else:
            sentence = sentence + " is"
        sentence = sentence + " touched."

        self.tts.say(sentence)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")

    args = parser.parse_args()
    try:
        # Initialize qi framework.
        connection_url = "tcp://" + args.ip + ":" + str(args.port)
        app = qi.Application(["ReactToTouch", "--qi-url=" + connection_url])
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) +".\n"
               "Please check your script arguments. Run with -h option for help.")
        sys.exit(1)
    react_to_touch = ReactToTouch(app)
    app.run()
```

## ALSonar

```python
#! /usr/bin/env python
# -*- encoding: UTF-8 -*-

"""Example: Use getData Method to Use Sonars Sensors"""

import qi
import argparse
import sys


def main(session):
    """
    This example uses the getData method to use sonars sensors.
    """
    # Get the services ALMemory and ALSonar.

    memory_service = session.service("ALMemory")
    sonar_service = session.service("ALSonar")

    # Subscribe to sonars, this will launch sonars (at hardware level)
    # and start data acquisition.
    sonar_service.subscribe("myApplication")

    # Now you can retrieve sonar data from ALMemory.
    # Get sonar left first echo (distance in meters to the first obstacle).
    memory_service.getData("Device/SubDeviceList/US/Left/Sensor/Value")

    # Same thing for right.
    memory_service.getData("Device/SubDeviceList/US/Right/Sensor/Value")

    # Unsubscribe from sonars, this will stop sonars (at hardware level)
    sonar_service.unsubscribe("myApplication")

    # Please read Sonar ALMemory keys section
    # if you want to know the other values you can get.


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
                        help="Robot IP address. On robot or Local Naoqi: use '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                        help="Naoqi port number")

    args = parser.parse_args()
    session = qi.Session()
    try:
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) +".\n"
               "Please check your script arguments. Run with -h option for help.")
        sys.exit(1)
    main(session)
```

Touch:

1. Session service using memory
    - self.memory = session.service("ALMemory")
2. Using memory key
    - touch = self.memory.getData("Device/SubDeviceList/RHand/Touch/Right/Sensor/Value")
      0.0 when false, 1.0 when True

Sonars:

sonar_front = self.memory.getData("Device/SubDeviceList/US/Left/Sensor/Value")

# Reinforcement Learning Tutorial:

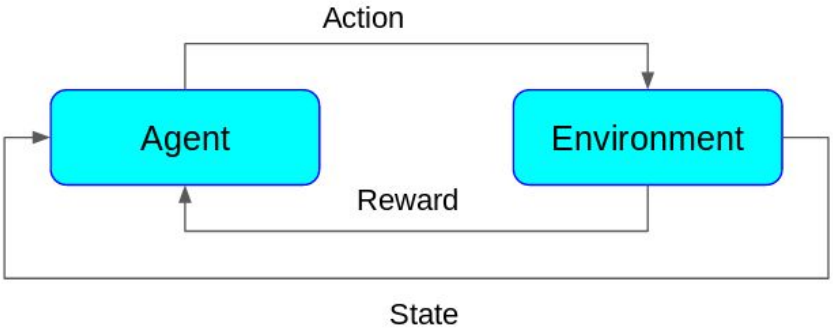
https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419

The idea behind Reinforcement Learning is that an agent will learn from the environment by interacting with it and receiving rewards for performing actions.

Learning from interaction with the environment comes from our natural experiences. Imagine you're a child in a living room. You see a fireplace, and you approach it.

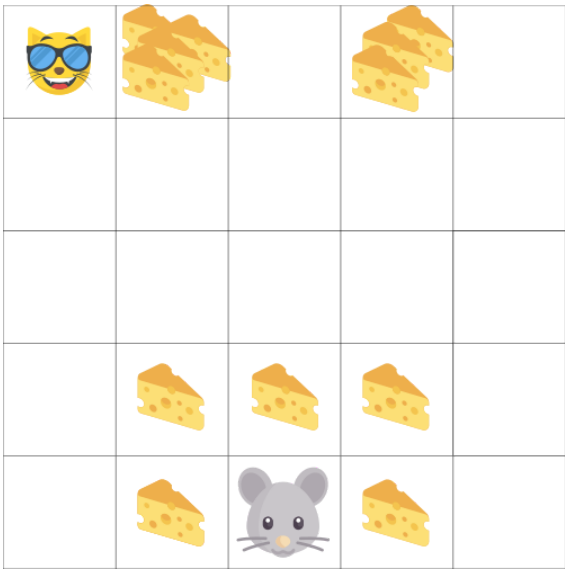

But then you try to touch the fire. Ouch! It burns your hand *(Negative reward -1)*.


**Reinforcement Learning Process**
The goal of the agent is to maximize the expected cumulative reward.




$$G_t = R_{t+1} + R_{t+2} + \dots$$

Other example



We define a **Discount rate** called gamma. It must be between 0 and 1.

- The larger the gamma, the smaller the discount. This means the learning agent cares more about the long term reward.
- On the other hand, the smaller the gamma, the bigger the discount. This means our agent cares more about the short term reward (the nearest cheese).

## Episodic or Continuing tasks

A task is an instance of a Reinforcement Learning problem. We can have two types of tasks: episodic and continuous.

### Episodic task

In this case, we have a starting point and an ending point **(a terminal state). This creates an episode**: a list of States, Actions, Rewards, and New States.

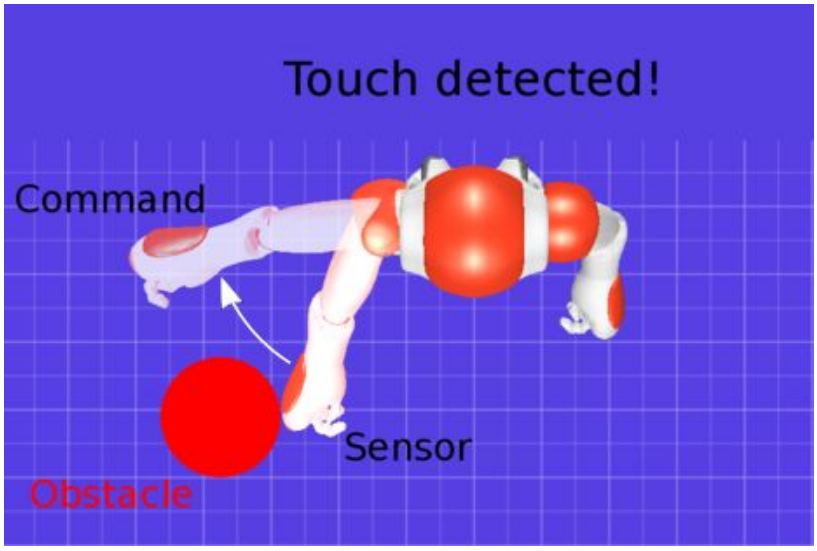
We have two ways of learning:

- Collecting the rewards **at the end of the episode** and then calculating the **maximum expected future reward**: *Monte Carlo Approach*
- Estimate **the rewards at each step**: *Temporal Difference Learning*

## Exploration/Exploitation trade off

- Exploration is finding more information about the environment.
- Exploitation is exploiting known information to maximize the reward.

What are the state, action, and reward in the following example?




Some libraries containing Reinforcement Learning methods:

https://pypi.org/project/pyqlearning/
https://gym.openai.com/

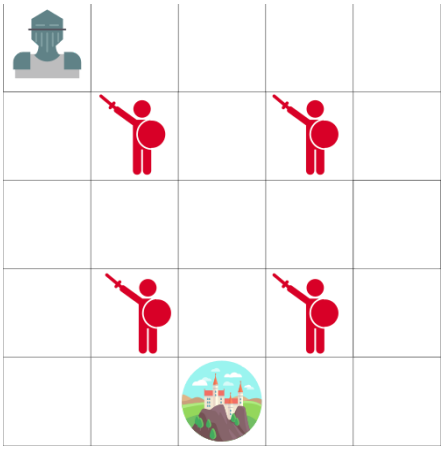
# QLearning

According to the Reinforcement Learning problem settings, Q-Learning is a kind of Temporal Difference learning(TD Learning) that can be considered as hybrid of Monte Carlo method and Dynamic Programming method. As Monte Carlo method, TD Learning algorithm can learn by experience without model of environment. And this learning algorithm is a functional extension of bootstrap method as Dynamic Programming Method. (pyqlearning)

https://github.com/simoninithomas/Deep_reinforcement_learning_Course/tree/master/Q%20learning/FrozenLake

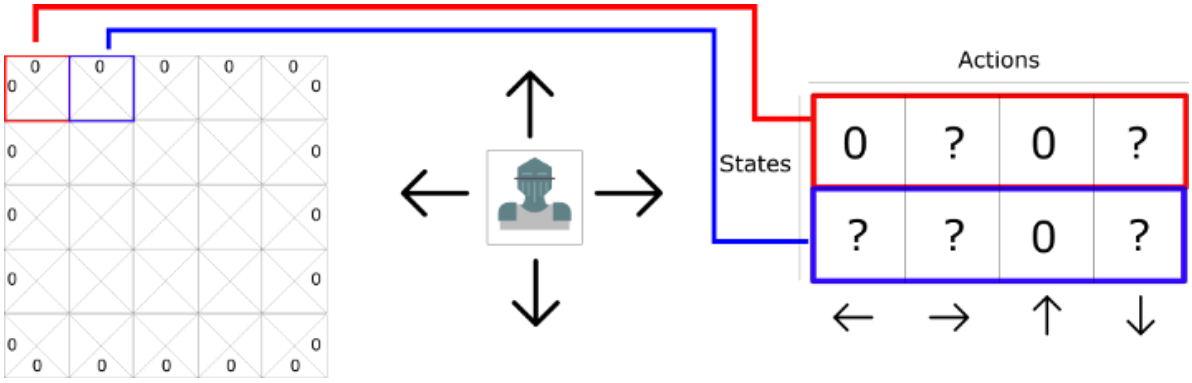
Q-Learning is a value-based Reinforcement Learning algorithm.

You can move one tile at a time. The enemy can't, but land on the same tile as the enemy, and you will die. Your goal is to go the castle by the fastest route possible



- You lose -1 at each step (losing points at each step helps our agent to be fast).
- If you touch an enemy, you lose -100 points, and the episode ends.
- If you are in the castle you win, you get +100 points.

Create a table where we'll calculate the maximum expected future reward, for each action at each state. This is called a **Q-table** ("Q" for "quality" of the action).




## Learning the Action Value Function

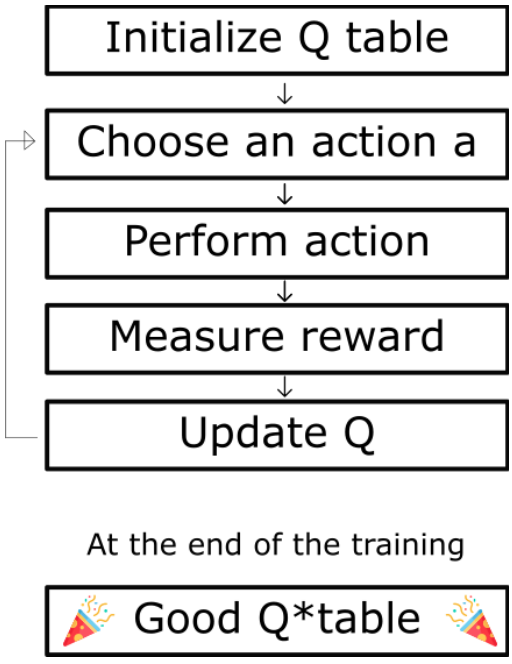
The Action Value Function (or "Q-function") takes two inputs: "state" and "action." It returns the expected future reward of that action at that state.

$$Q^{\pi}(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | s_t, a_t]$$

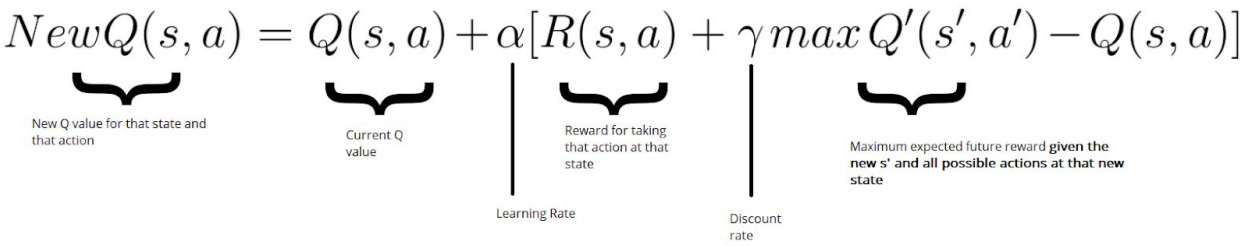
Q value for that state given that action

Expected discounted cumulative reward ...

given that state and that action



**Bellman equation (QLearning)**:




# Learning a Behavior

**Looking for a person to handshake saying hello**

The robot have 4 basic actions for follow a person, rotate left, rotate right, and extend the arm while saying hello.
Sensors: Sonars (something in front), HandTouch, FaceDetection.

With these actions and sensors, Nao will generate a complex behavior for looking for a person and extending the arm and saying hello. The goal of the robot is to sense a touch on its hand. In order to learn, it needs at least one person in front of him giving the rewards (touching the hand).

Here we use reinforcement learning (QLearning) to create such behavior.

Task:
A code of QLearning using some of the NAOqi modules, presented here, will be provided. There will be some missing parts to be completed. Also, a live demo of the trained and learned behaviors will be shown.

**Requisites:**
NAOqi
https://community.ald.softbankrobotics.com/en/resources/software/language/en-gb

**http://doc.aldebaran.com/2-5/dev/python/install_guide.html#python-install-guide**