

# **PROJECT ATHENA**

## **DOCUMENTATION**

# Table of Contents

<b>Introduction</b>	3
<b>Github</b>	4
<b>Versions</b>	5
<b>Gameplay</b>	7
<b>Functions</b>	8
Green Path	9
<b>Heroku / MongoDB</b>	11
How MongoDB works	11
Access the URI	12
Resetting the Database	14
<b>Possible Bugs / Oddities</b>	17
<b>Setting up Choregraphe</b>	18
Installing Choregraphe Suite	18
Installing Python SDK	18
<b>Working with NAO</b>	20
Moving NAO Outside of Choregraphe	20
<b>Python Server Setup</b>	21
Setting up Websockets in Choregraphe	21
Connecting Heroku to Server	21
Connecting to the CRS Lab Router	21

# Introduction

Welcome to Project Athena. This document is here to help you understand how the game simulations work, how to navigate the files, how to use the database, and many more aspects of the project.

The aim of Project Athena is to create a framework for testing human-robot interaction hypotheses. In this project, we created a first response domain, where a human-robot team is assigned to a first-response task after a disaster occurred. In this scenario, the human's task is to team up with a remote robot that is working on the disaster scene. The team goal is to search all the marked locations as fast as possible and the human's role is to help the robot by providing high-level guidance as to which marked location to visit next. The human peer has access to the floor plan of the scene before the disaster. However, some paths may be blocked due to the disaster that the human may not know about; the robot, however, can use its sensors to detect these changes. Due to these changes in the environment, the robot might not take the expected paths of the human.

There are three versions of the game simulation. The first is Online Explanation Generation[1] which allows the human to receive model modifications from the robot simultaneous to plan execution toward the target goal such as "Getting to the room requires breaking a hidden obstacle". Arbitrary Explanation Generation functions the same but with the explanations provided at random times. The last version is Minimally Complete Explanation[2], where the robot explains to the human the minimum required explanations for the entire goal to make sense for the human as a chunk. Click [here](#) to see more about the game simulation versions.

In this project, an actual robot is connected to the simulation through a server so it can mimic the movements made by the simulated robot in a real mock-version of the disaster scene.

For debugging and learning what the game is doing, you can use the Developer Tools in browser. On Chrome, Hit CTRL+SHIFT+I or click the three dots, hover over More tools, and click on Developer Tools. On Firefox, Hit CTRL+SHIFT+I or click on the three bars, click on Web Developer, then click on Toggle Tools. The console tab is the most useful for this project. It will show errors in red and warnings in yellow. Updates from the game such as the robot's current battery level, number of tools, and other information. Whatever variable you need to debug can be printed here by adding something like `console.log("Variable is now: ", varName);` into the code. Anything you do not need anymore can simply be commented out or removed.

# Github

It is important to keep the repositories private. With a free Github account, up to 3 collaborators are allowed on a private repository. If you want more collaborators, Github Pro is required. With ASU, you are eligible to receive GitHub Pro for free for a year. Click [here](#) to set that up. You need verification such as a picture of ASU ID or Unofficial Transcript or picture of your current semester schedule. In a few days you should get a response from GitHub. They may deny your request and you may have to send in a different form of verification. There are a few different repositories in GitHub, which is explained in the versions sections below.

# Versions

There are three main versions of the game simulation. There are some older legacy versions as well. Use the chart below to become familiar with the versions

Version Name	What it is	Heroku Link
Online_Explanation_Generation	The most basic version of the game. Click <a href="#">here</a> to learn about how general gameplay works, then you can understand the other versions.	<a href="https://project-athena.herokuapp.com/">https://project-athena.herokuapp.com/</a>
Random_Explanation_Generation	Same as the Online_Explanation_Generation version, but with explanations given in random order. The purpose is for use of null hypothesis for research purposes	<a href="https://project-athena-random.herokuapp.com/">https://project-athena-random.herokuapp.com/</a>
Minimally_Complete_Explanation	Same as the basic game but does not stop the explanation at all the points of interest (tools, batteries, etc). Rather, it gives all the explanations at one time for the entire move sequence. It asked the user if it makes sense, and after the user answers the robots proceeds.	<a href="https://project-athena-npause.herokuapp.com/">https://project-athena-npause.herokuapp.com/</a>
Game-1-Test	Original version that we built off of. We did not change anything from Game 1 Training or Game 2. After a few sprints, we left this one alone and coded exclusively in Online_Explanation_Generation since we changed a large portion of the game structure.	N/A
Game-1-Training	Same as Game-1-Test, but asks the user the question "Do you think the robot is doing the right thing?" For every move.	N/A
Game-2-Test	In Game 2, the user need not choose which rooms to go to.	N/A

	The program automatically chooses the optimal path and the robot goes to every room. With this version, the user can just sit and watch.	
Game-2-Training	Same as Game-2-Test, but asks the user the question “Do you think the robot is doing the right thing?” For every move.	N/A

[This](#) is the link to the original game hosted on Heroku that we were given to build off. This was built off of Game1-Training-Master. You can play it and get a sense for how much work has been put into the game from this until the most recent versions.

# Gameplay

To play the game, first read through the tutorial to understand the instructions that are given to the user. Here we will explain the technicalities. You start with the robot (represented by the red square with an arrow) in a corner. The green squares represent someone in need of medical attention. Click on those to go there. Any order is fine, the game is made to work for any given situation. When you click on one of the goals, the game first shows a green path that the robot will take. This is to help the human understand what the robot's next move will be as sometimes it may be in a different direction. The robot will take a longer path if it needs a tool or battery recharge. It may also be avoiding obstacles that are hidden but do exist on the map and are not visible to you at the moment. Depending on the simulation version, explanations will be given to the user to the right of the map either all at once, progressively, or randomly. Could you please add the differences of the gameplay in terms of different versions of the game? It would be helpful if you show those version differences in terms of figures.

In order for you to complete the next move, you must answer the question located at the bottom of the game: "Did you expect this move from the robot?" as yes or no. It doesn't matter what you answer as this does not affect the robot's decision. The BFS Function has already found the optimal path and is set on taking that path. Nothing can disrupt that.

As you play the game, you will pick up tools which are necessary to break through obstacles. Some obstacles are visible (the blocks look like a cracked wall). Hidden obstacles become visible when you are within 1 block of the hidden obstacle. They will blink green while you are at this close distance to them. When you exceed that distance they stop blinking. When you pick up a tool or tools, you can check the inventory at the top above the explanations. It displays a tool icon with 1 number. When you break a hidden obstacle, the number is updated in addition with the explanation. You will also notice to the left of the tool the battery. It starts at full represented by 4 green blocks. Moving one blocks takes a little battery and breaking and obstacle takes significantly more battery power. Picking up tools and battery packs have no effect. You will notice it change color and eventually come to one 1 block. The BFS function knows the current battery level and how much will be taken away for possible moves, so it takes this into account. So the BFS prevents the robot from running out of battery. If it will reach below 25% within a move, it will include a redirect to a battery pack on the map. All battery packs and tools are hidden, but the game is easily modified so you can see them. Thus you can always know the status of the robot.



Fig. 1 1 Tool displayed if the user has only 1 tool left

Immediately after reaching the last goal, the page is redirected to the questions page. This is a short and simple survey for the user to answer. Answers are stored into a database and a unique Hex Code is generated for use of authenticating that Amazon Mechanical Turk users did in fact complete the entire game.

# Functions

- Function `setMap()`- this function pulls a random map from a premade list and returns it to be displayed on the website. In the final three versions, it only initializes the map. Custom maps of any  $n \times n$  size can be made using `maps.js`. We only have one map currently - which is the one we decided on. A random function is used to select the map. If you want more maps, just add more into `maps.js`. Numbers from 0 to 24 represent different types of tiles. The key is found in the variable *floortypes* inside `game.js`.
- Function `Character()`- this is the function that contains all the information for the users current state and destinations, it also holds the functions for moving the robot around.
- Sub-Function `Character.prototype.placeAt()` - This function is used in our case to change the tiles green and display hidden tiles when showing the green path. It will also push explanations as the green path is displayed. If not green path, the function just uses the real position.
- Sub-Function `Character.prototype.processMovement()` - This function originally was used for animating the robot move from tile to tile. In the final three versions it is bypassed in some cases to make the robot move faster.
- Sub-Function `Character.prototype.canMoveTo()` - This function checks if the `x,y` are not walls or out of bounds
- Sub-Function `Character.prototype.moveEligible()` - This function checks if the tile is eligible to move onto (future improvement: make `canMoveTo` and `moveEligible` the same since they do the same thing)
- Sub-Function `Character.prototype.resetTarget()` - this Function is used to set certain values to a preset value before continuing. The delay move is arbitrarily small (1000 is the normal value) so as to make it move fast. (be careful to not make it too small, it will glitch in some cases)
- Sub-Function `Character.prototype.moveToLocation()` - This function contains the checks for each movement and call a different function to do the movement itself. Since javascript is a non-sequential language, this is where the `Wait_For_Click` is used to wait for the user to respond to the prompt of screen.
- Sub-Function `Character.prototype.confirmLastMove()` - This function will display a prompt to the user.
- Function `toIndex()`- this returns the position of the `x` and `y` in the program



- Function `getTileType()`- this returns what the tile is at that position. This tells the BFS if this tile is a tool, wall, battery, etc.
- Function `confirmAction()`- returns the confirmation of whether or not the user thinks it's doing the right thing.
- Function `isAllVisited()`- this checks if the map has everything already picked up
- Function `BFS()`- this is the algorithm that uses the breadth first search algorithm.
- Function `TileBFS()`- this will look for the different tiles specified as target1,2,3,4. It is identical to BFS aside from having four targets.
- Function `reducedIf()`- this is a function that checks each thing around the robot and returns whether or not you could go that direction
- Function `adjacentEdges()`- this is using the `reducedIf()` to find out if the paths could be taken
- Function `drawGame()`- this is the function that does the animation and the update for the map.
- Function `displayMessage()`- this is the function that allows the javascript to print to the textbox.
- Function `calcBattLVL()`- this function checks the value of total percent counter and then sets the current battery level. This function is called at many points throughout the game, possibly too many times.

## Green Path

Green path works by using the same process as a normal move in the simulation, but will put a slight tweak to how it operates. It starts with when the target is selected, it will use the BFS function to find the shortest path. This path is passed to the functions that process movement like `moveToLocation`. Prior to the movements being enacted, the simulation saves the previous map state, robot location, stats of the robot like battery level and tools collected. It will then go along the path found earlier and replace every tile that its been on with a green tile. This movement is able to be speed up since its not using an animation to move the robot from tile to tile and thus, being able to process faster. When it lands on a panel that it wants to display, it will store the tile its on, turn it green, and later change it to a displayable tile. This was a quick work around a problem earlier. Once the green path reaches the goal, and the user confirms the prompt, the saved data from before is used to reset the stats, position, and map to its original state. Everything is done again to make it move but without the green path traits. In

future updates it could be shortened so that the beginning process does not have to happen again.

Bugs that were “fixed” for green path:

- tileTo and tileFrom have to be stored and replaced when going back to real movements since one of the movement functions uses that to move the robot.

# Heroku / MongoDB

## How MongoDB works

MongoDB is the database used to store all the data obtained from the simulation. Data is sent to MongoDB once all sub-goals are reached. This occurs in the `moveToLocation` function. More specifically, in the `if(isAllVisited(gameMap))` if statement. Currently, the simulation will send the duration of the game, map id, number of yes's and no's clicked, number of times that the user understood or did not understand either the greenpath or the explanation, the number of actions performed by the robot the time taken by each user decision and the avg of these times to the database.

If you would like to send additional information the code in the aforementioned if statement will have to be changed.

```
450 $.post("/game", {
451     gameTime: timePassed,
452     mapId: mapId,
453     numYes: yesClicked,
454     numNo: noClicked,
455     pathYes: yesClickPath,
456     pathNo: noClickPath,
457     expYes: yesClickExp,
458     expNo: noClickExp,
459     gameActions: numofActions,
460     timeArray: storeTime,
461     decTime: timeAvg}, data =>
462 {
463     sessionStorage.setItem('explainabilityUUID', data);
464     window.location.href = '/questions';
465 });
```

Let's say that you wanted to also send the number of obstacles the robot broke during the simulation and you store this number in the variable "brokenObstacles".

To send this piece of data you would have to do the following (marked in green):

```
...
459 gameActions: numofActions,
460 timeArray: storeTime,
461 decTime: timeAvg,
462 numOfBrokenObs: brokenObstacles}, data =>
...
```

Where "numOfBrokenObs" is an arbitrary name that you can choose.

Once the above is completed, you will have to modify the `index.js` file that is in the main folder of whichever game version you are working on.

The index.js file is the one that takes care of communication between MongoDB and Heroku application.

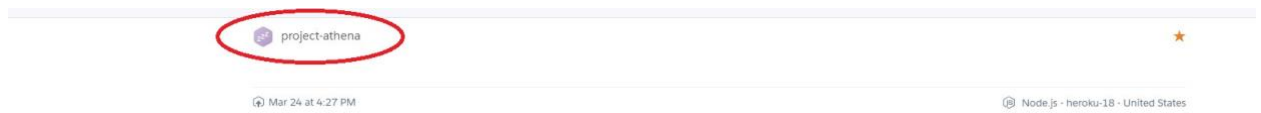
The following line is the one that connects the application to a specific MongoDB database:

```
mongodb.MongoClient.connect(process.env.MONGODB_URI ||  
"mongodb://<athena>:<athena123>@ds151943.mlab.com:51943/heroku_1vs9p2ck", f  
unction(err, client) .....
```

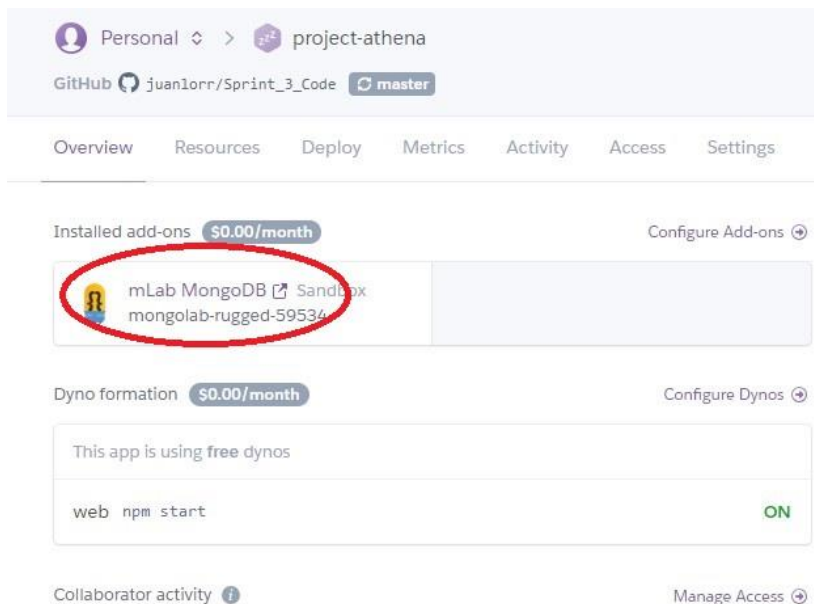
The section marked in red is the uniform resource identifier (URI) use by the database that the data will be sent to. It can be accessed in the following way:

## Access the URI

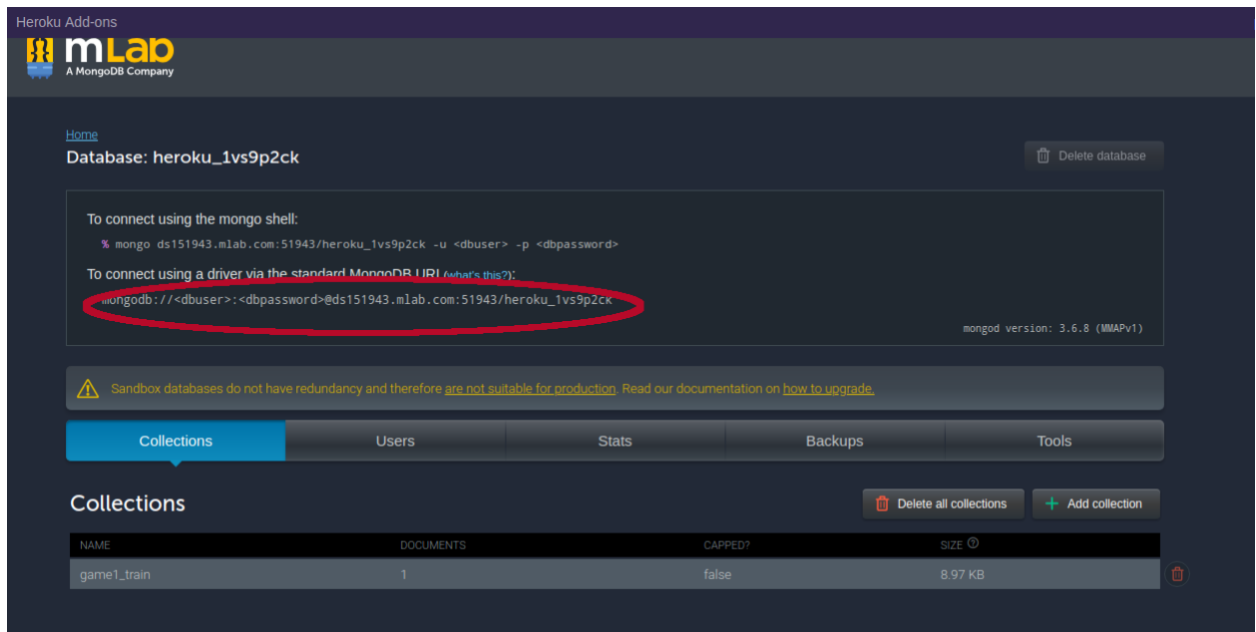
1. Go to <https://dashboard.heroku.com/apps>
2. Click on the version whose URI you want to access



3. Click on mLab MongoDB



4. The URI is shown in the image below.



5. All three versions of the game have dbuser = athena and dbpassword = athena123. If you would like to change this you will have to create another user in the “Users” tab (see picture above)

In the index file, you will have to add the following to line 61 (the line number may change so just follow the line numbering in the image below)

```
60 |  
61 |   const obj = {id: id, gameTime: req.body.gameTime, mapId: req.body.mapId, numYes: req.body.numYes};  
62 |   users[id] = obj;  
63 |   res.send(id);
```

```
const obj = {id: id, . . . . . , decTime: req.body.decTime,  
numOfBrokenObs:req.body.numOfBrokenObs};
```

The obj variable must be modified one more time before being sent to the database:

```

71     const obj = {
72         id: req.body.id,
73         gameTime: users[req.body.id].gameTime,
74         mapId: users[req.body.id].mapId,
75         numYes: users[req.body.id].numYes,
76         numNo: users[req.body.id].numNo,
77         Num_of_Yes_click_path: users[req.body.id].pathYes,
78         Num_of_No_click_path: users[req.body.id].pathNo,
79         Num_of_Yes_click_exp: users[req.body.id].expYes,
80         Num_of_No_click_exp: users[req.body.id].expNo,
81         gameActions: users[req.body.id].gameActions,
82         timeArray: users[req.body.id].timeArray,
83         timeAvg: users[req.body.id].decTime,
84         answers: req.body.answers};

```

The following must be added to the statement above:

```

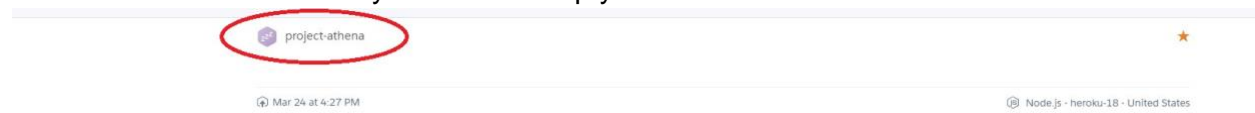
. . .
timeAvg: users[req.body.id].decTime,
numOfBrokenObs: users[req.body.id].numOfBrokenObs,
answers: req.body.answers};

```

## Resetting the Database

Follow these steps to reset the database:

1. Go to <https://dashboard.heroku.com/apps>
2. Click on the version you need to empty



3. Click on mLab MongoDB

Personal > project-athena

GitHub juanlorr/Sprint\_3\_Code master

Overview Resources Deploy Metrics Activity Access Settings

Installed add-ons **\$0.00/month** [Configure Add-ons](#)

mLab MongoDB Sandbox  
mongolab-rugged-59534

Dyno formation **\$0.00/month** [Configure Dynos](#)

This app is using free dynos

web npm start **ON**

Collaborator activity [Manage Access](#)

4. Click on the name of the collection

mLab  
A MongoDB Company

[Home](#)  
Database: heroku\_1vs9p2ck [Delete database](#)

To connect using the mongo shell:  
% mongo ds151943.mlab.com:51943/heroku\_1vs9p2ck -u <dbuser> -p <dbpassword>  
To connect using a driver via the standard MongoDB URI ([what's this?](#)):  
mongodb://<dbuser>:<dbpassword>@ds151943.mlab.com:51943/heroku\_1vs9p2ck  
mongodb version: 3.6.8 (MMAPv1)

Sandbox databases do not have redundancy and therefore are not suitable for production. Read our documentation on [how to upgrade](#).

[Collections](#) [Users](#) [Stats](#) [Backups](#) [Tools](#)

Collections [Delete all collections](#) [Add collection](#)


NAME	DOCUMENTS	CAPPED?	SIZE
game1_train	1	false	8.97 KB

System Collections

NAME	DOCUMENTS	SIZE
system.indexes	1	0.11 KB

5. To delete everything, click delete all documents in this collection. To delete an individual record, click the trash can next to the record you wish to delete

Home : { db : "heroku\_1vs9p2ck" }


Collection: game1\_train 

Documents Indexes Stats Tools

Documents

— Start new search — ▾

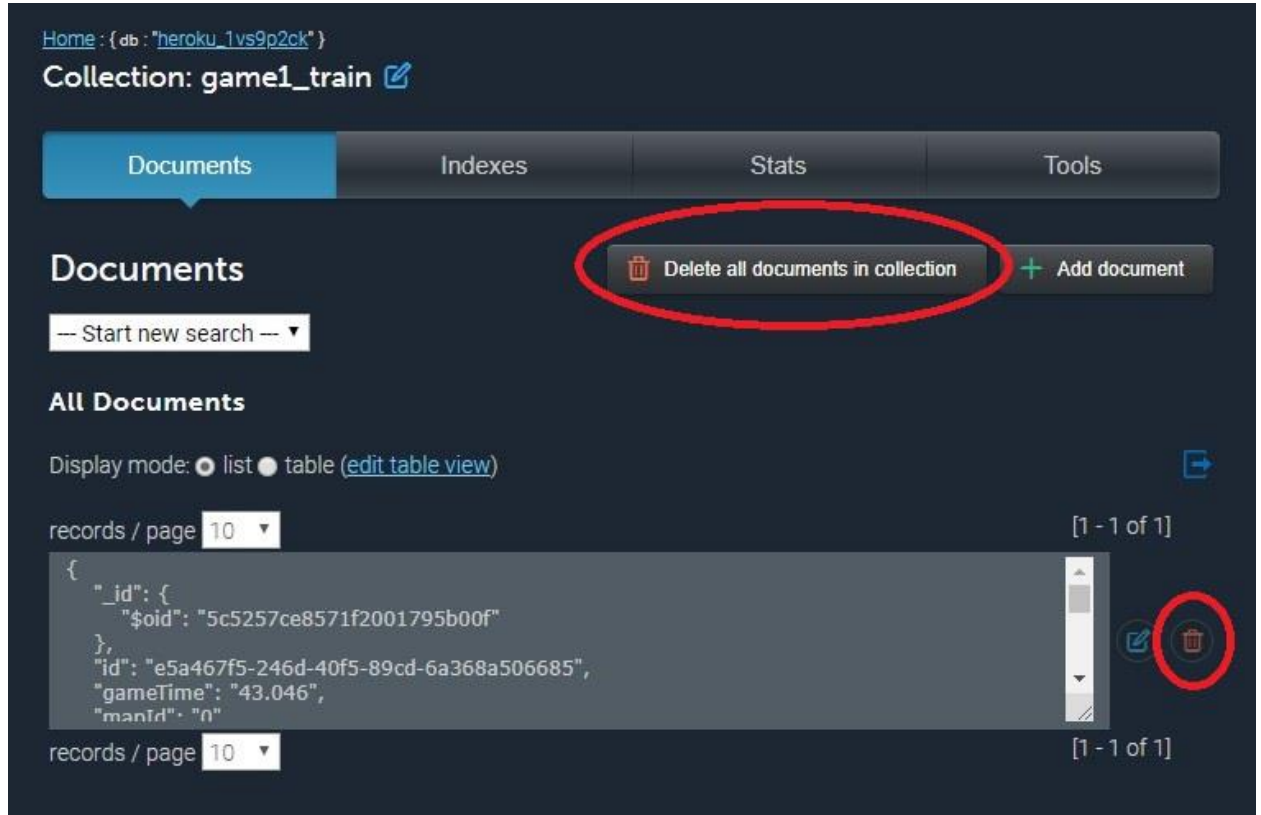
All Documents

Display mode: ☒ list ☐ table ([edit table view](#)) 

records / page 10 ▾ [1 - 1 of 1]

```
{
  "_id": {
    "$oid": "5c5257ce8571f2001795b00f"
  },
  "id": "e5a467f5-246d-40f5-89cd-6a368a506685",
  "gameTime": "43.046",
  "manId": "0"
}
```

records / page 10 ▾ [1 - 1 of 1]





# Possible Bugs / Oddities

Below are some bugs that we have encountered that may or may not pop up again. Also included are just some odd things that just happen so you can be aware to look out for.

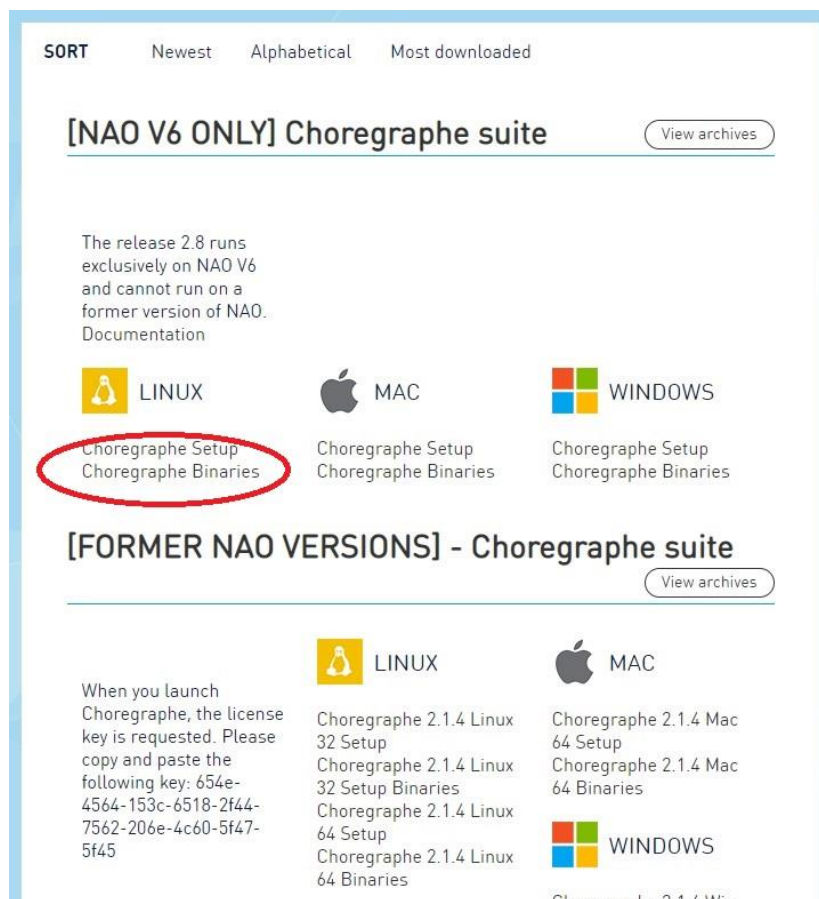
- Public folder - You may notice the public folder that is inside each game simulation folder. Inside the public folder are all the image sprites, game.js, maps.js, and tutorialgame.js. Leave those files in there. Everything breaks without this. For hosting on Heroku, we had to add some of these files here. To test locally, use the file structure used in legacy versions.
- Firefox issues - we have noticed that sometimes Firefox has some issues with the game simulation. Sometimes bugs happen that don't on Chrome, and sometimes the game simply won't load.
- Clicking very fast - We have noticed in the past that sometimes clicking the yes and no buttons extremely fast may cause problems. This may be very rare because we solved the problem by graying out the buttons until the robot has come to a stop and needs further communication with the human. This is just something to look out for. Without the graying out implemented, the robot will go through walls, teleport, and even break outside of the game border!

# Setting up Choregraphe

Choregraphe works on Windows 10, Mac, and Linux. Ubuntu 16.04 or 18.04 is the best choice. Note, however, that Ubuntu 18.04 is the only version that works with the server code as far as we know, so we highly recommend only using Ubuntu 18.04.

## Installing Choregraphe Suite

1. Go to [this link](#) to download Choregraphe:
2. Use these login credentials:  
Username: crslab.asuai@gmail.com  
Password: Humans@robots
3. The version you want is: **2.8.5 (We have NAO V6)**
4. For Ubuntu, it is easiest to download the binaries



## Installing Python SDK

**Note: This is needed in order for the server code to work**

1. Go to [this link](#) to download Choregraphe:


2. Use these login credentials:  
Username: crslab.asuai@gmail.com  
Password: Humans@robots
3. The version you want is: **Python 2.7 SDK 2.1.4.13**


**SORT**    Newest    Alphabetical    Most downloaded


## [NAO V6 ONLY] SDK

[View archives](#)

The release 2.8 runs exclusively on NAO V6 and cannot run on a former version of NAO.  
Python SDK installation guide  
C++ SDK installation guide


 **LINUX**  
Python SDK  
C++ SDK  
Cross Toolchain


 **MAC**  
Python SDK  
C++ SDK


 **WINDOWS**  
Python SDK  
C++ SDK

## [FORMER NAO VERSIONS] - Python NAOqi SDK

[View archives](#)

 **LINUX**  
Python 2.7 SDK 2.1.4 Linux 32  
**Python 2.7 SDK 2.1.4 Linux 64**

 **MAC**  
Python 2.7 SDK 2.1.4 Mac 64

 **WINDOWS**  
Python 2.7 SDK 2.1.4 Win 32 Binaries  
Python 2.7 SDK 2.1.4 Win 32 Setup

# Working with NAO

Inside Choregraphe, there are many sample moves and other functions preprogrammed. You can piece these pieces together into a flow diagram. You can control the variables and timing.

NAO has a lot of documentation online on the Aldebaran website. Here is a link to several different walking motions: <http://doc.aldebaran.com/1-14/dev/python/examples/motion/walk.html>. On the left side you can see the table of contents and a search. With those you can find lots of code examples and a little bit of explanation.

## Moving NAO Outside of Choregraphe

Using our code provided, you can see an example of how we can control NAO outside of Choregraphe just using python code in Movefunctions.py from the Server\_Python repository.

# Python Server Setup

## Setting up Websockets in Choregraphe

- Download choreograph for on your system with Ubuntu 18.04  
[http://doc.aldebaran.com/2-4/dev/community\\_software.html](http://doc.aldebaran.com/2-4/dev/community_software.html) more instructions [here](#)
- Download python-2.7 and the python\_2.7-naoqi [http://doc.aldebaran.com/2-1/dev/python/install\\_guide.html](http://doc.aldebaran.com/2-1/dev/python/install_guide.html) more instructions [here](#)
- Pip install websocket-server
- In the terminal run “export LD\_LIBRARY\_PATH=\${LD\_LIBRARY\_PATH}:path/to/choreograph-suite/lib”
- Also in the terminal run “export PYTHONPATH=\${PYTHONPATH}:path/to/python-sdk”
- **(These instructions are incomplete and might need further troubleshooting to get the server to work)**

## Connecting Heroku to Server

1. Login to router and port forward (we used 9001 tcp/udp)
2. Open port on servers computer, Steps below:
  - a. sudo ufw enable
  - b. sudo ufw default deny
  - c. sudo ufw allow 9001 (just a port we used that was not being used)
  - d. sudo ufw status verbose

Heroku files websocket must connect to the public IP of the server's computer. (Google what's my ip and look for public ipv6). When running the server over heroku use ws instead of wss. When running heroku, make sure the web server is allowing for unsecure connections.

\*Make sure to run the server first by typing in server\_v2.py, then start the game at <https://project-athena-npause.herokuapp.com/game>.

## Connecting to the CRS Lab Router

- Disconnect from ASU Network
- Connect to CRS LAB network
- Password: **CRSLAB511ASU**