Bodacious Battle Bots

TECHINICAL

MANUAL

Technical Manual

Table of Contents

1. Programming Language….......//fix page #............GABI PLEASE DO THIS TOO.................................................2

**Programming Language and Structure of Files**

Programming Language

The programming language used throughout all the skills is JavaScript. Misty is capable of running only JavaScript and C#, we chose to use JavaScript since C# is still under development.

Structure of Files

The “root” directory is MistySkills. Inside there, there is a folder for every skill, and the name of the folder is the name of the skill.

Inside each folder of a skill, you will find at least three files. The source code, this file will have the .js extension. The JSON file, which will have the .json extension, this file holds the parameters needed for the skill, such as identifiers, variables, and instructions of how the robot should interact with that skill. Also, you will find a read.me, which gives a brief overview of the skill along with some details.

Some skills will have more than three files inside. These files are used during the skill and will only be images or audios.

Coding Practices

One of the most important coding practices involved in this project that is different than usual is the fact that if you want to use a function in a RegisterTimerEvent method, you will need to name as follow “ \_nameOfFuncion()” (underscore plus the name of your function).

Another important practice that in this project is different than others is how to declare global variables. In order to declare a global variable you will need to use misty.Set(), which takes three parameters, the first one is the name of the variable (string), the second one is the value, and the last one is if it should be kept after the skill is done. In order to access this variable, you will need to use misty.Get(), which requires only one argument, which is the name of the variable (string).

UML Use Case Diagram

Below is the Use case diagram. The operator of the Misty Robots will interact with the dashboard, which will then interact with the robot, and finally with the audience/other robot.

Diagram

Description automatically generated

Class Diagram

Bellow, the class diagram with a more specific view of how the interactions are being done and expected for the project run smoothly.

Diagram

Description automatically generated

**Brief introduction to skills**

Dancing  
In this skill, Misty moves its arms and head. It also turns around and plays an audio.   
  
Detecting Objects  
Misty starts to walk while it detects objects. It plays and audio when the object is found.  
  
Text to Sound  
The robot gets a string and uses an API to speak the text in the desired language.  
  
Sound to Text  
The robot hears a phrase and uses an API to turn that spoken phrase into text in the desired language.  
  
Conversation  
Misty answers to some audio inputs based on previously coded interactions.

Translation  
The robot hears a phrase, transforms it to a string, then translates the string to another language, and then gets the string and transforms it in audio. All that using APIs.

Tic-tac-toe  
Misty plays tic tac toe on a board that animates Xs and Os for player one and player 2 until the game ends in a win lose or draw. Player one and two’s turns are made within the skill itself.

Security Camera  
Misty starts streaming and it is possible to see the video streamed using a website.

Reaction  
Robot copies the arms movements that the person does, movements possible: arms up and down.

Face Recognition  
Misty recognizes faces and displays and audio each time it recognizes the person. The audio changes depending on the amount of interactions Misty had with that person.

Rock Paper Scissors  
The robot plays rock paper scissors.

Start with IP Address

This skill runs on startup. The only thing that it does is show the IP address of the robot on its screen once you tap on the robot’s chin.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Special details of the skills code**

Dancing  
In this skill, Misty moves its arms, head. It also turns around and plays an audio.

We set Misty’s face to admiration, that we got from the Misty SDK website.

We created two functions to handle the arms movements. In these functions, we change the position of the arms.

In the head movement function, Misty turns its head left and right by us just changing the second parameter of the MoveHead() function.

In the turning function, Misty turns back and forth in a preset circle.

The most important part of this skill is handling the loops. Misty does not allow any kind of while loop inside her code. The way the robot uses loops is through registering an event to happen every X amount of time.

We used the RegisterTimerEvent to achieve that. This method needs 3 parameters: first, the name of the function (without the underscore) as a string, second, after how many milliseconds should it be triggered as an integer, and third, if it should repeat or not as a Boolean.

For this skill, there is nothing different in the JSON file from what the Misty SDK website generates.

Detecting Objects  
In this skill misty will walk, avoiding hazards and when the object is found she plays an audio saying that the object was found.

There are two major functions in this skill, the first one is \_object\_detection(). If the robot detects either a chair/laptop/cup/backpack it will stop moving and it will display a different “emotion” and play a different sound as a reaction of finding these objects.

The second major function is Hazard(). Misty starts the program driving, when there is something blocking its way, it triggers Misty’s sensors and gets data from each one of them. When at least one of the sensors is triggered because of an obstacle in the way, the robot tries to find a route without any hazards, changing the direction that the robot will drive next.

For this skill, there is nothing different in the JSON file from what the Misty SDK website generates.  
  
Text to Sound  
Straight forward skill, transforms a string into an audio that Misty will play.

We have 3 functions speakTheText, \_Base64In, and \_speakTheText.

The speakTheText method takes only one parameter. This parameter is a string and should be the text that you want Misty to Speak. Inside this method we have an important setup to do in order to run this skill. We have to use JSON.stringify() to make a string out of the arguments we need to use. These arguments are passed to the Google API which then sends us the audio response.

The most important parameters under the arguments are ‘input’, which should be the only parameter used in this method, ‘languageCode’, which is the language of the input, and ‘audioEncoding’, which should always be “LINEAR16” for the rest of the code to work.

After setting up, you can then use SendExternalRequest to talk to the Google API.

The \_Base64In function is where the Google API talks to misty and saves the audio file, then plays it. This method needs to start with an underscore, because it is being called from SendExternalRequest and getting data from it.

The \_speakTheText function is only to allow the Google API to talk to misty, by registering an event that misty will be waiting before running the rest.

Our JSON file in this skill plays a very important part and have a different set up. Inside the “parameters”, there is a variable called "APIKEY\_Google" which is followed by a colon and then a string. This string is the key that you should get from google. Explanation on how to get this key is under “Installation and Requirements to work with This Project”.

Sound to Text

Record an audio, then displays it on Misty’s screen.

There are 2 functions that we created \_voice\_record\_complete\_message and register\_voice\_record\_complete.

In order to accomplish this task, we used the built-in function CaptureSpeechGoogle. Which takes the following parameters: overwriteExisting(which should always be false), silenceTimeout(how long should Misty wait without hearing anything before it moves on), maxSpeechLength(how long should it record if the silenceTimeout does not stop it), requireKeyPhrase (if misty should wait for a “Hey, Misty” before it starts recording), captureFile(should always be true), speechRecognitionLanguage (the language that she is going to hear in language code), key (Google API key, that should be inside JSON file).

Since the Google API will return a lot of data, we need to specify what we want to use. For this, we used the register\_voice\_record\_complete function. Which add the proper return cases that we should look from the API, then register the event voice\_record\_complete\_message.

The \_voice\_record\_complete\_message is an event; therefore it needs to begin with an underscore. Inside, we just call built in functions to display the proper results.

Our JSON file in this skill plays a very important part and have a different set up. Inside the “parameters”, there is a variable called "APIKEY\_Google" which is followed by a colon and then a string. This string is the key that you should get from google. Explanation on how to get this key is under “Installation and Requirements to work with This Project”.  
  
Conversation

Executing this skill will allow the person engaging Misty to have a somewhat varied verbal conversation for a few sentences. If Misty doesn’t recognize an utterance by the audio she listens to, she will play audio messages that state it didn’t compute the utterance heard. It then starts to listen for incoming audio for up to seven seconds. The utterances length in time can vary from a few seconds all the way up to seven. After around five seconds Misty will automatically end the API call if no audio is read in. In this case, Misty will still play audio stating that it doesn’t recognize what you said. Since the accepted utterances from the user to the robot are hard coded, the user must say something she will recognize (recognized utterances can be seen in the source code. The conversation was then supposed to incorporate the dashboard to socialize with another Misty.

**Functions:**

Function **\_Convo**(data){} – this function is used to initialize the first part of the conversation with the user. After Misty greets the user, it calls the CaptureSpeechGoogle API request to take in speech then convert it into text. If the first three accepted utterances aren’t heard, the user will be asked for utterance until they say something Misty recognizes.

Function **Start\_The\_Convo**() – This function calls the misty.RegisterEvent(“Convo”,”VoiceRecord”, 100,true) which executes the \_Convo function. This function will be called to restart \_Convo each time the user’s utterance isn’t recognized. This allows the API to be called for each iteration of \_Convo so it can check if the utterance is acceptable.

Function **\_MainConversation**() – This function takes place after the initial conversation successfully starts. Global variables that hold Misty’s two responses and the two accepted user utterances are manually set to the values defined within the switch case statement. After the function executes a random response from the switch case statement, it then calls the speakTheText() function to respond to the initial utterance which calls the text to speech API which plays the inputted string’s audio equivalent. Start\_The\_Convo2() is called where it listens to incoming audio once more.

Function **Start\_The\_Convo2**() – Like the main conversation function, this function simply continuous the second part of the conversation. Misty.RegisterEvent(“Convo2”…) is called to execute the \_Convo2() function which

Function \_**Convo2**( ) - The values of the global variables that were set in the \_MainConversation() function and passes them into the \_LoopConversation function.

Function \_**LoopConversation**(, , ,) – This function has three parameters it accepts which includes both accepted utterances of a particular response and the response text to speak to the user calling the text-to-speech API. The conditional statements measure rather or not the user’s utterance at the final stage of the conversation is recognized. If it isn’t, ask the user to speak their response again. Otherwise, \_IsReady( , ) is called to have Misty speak one last response based off the two recognized utterances that may have been inputted. It then Unregisters the event which started “Convo2”.

Function **similar**( , ) – calculates the percentage accuracy of a string to another string value. This is primarily used to compare what the user speaks and the two acceptable phrases at several parts of the conversation. The utterance won’t be accepted if it doesn’t return a value above 65%.

Function **speakTheText**() – accepts a string which is sent to the text-to-speech API. The data sent to the API is then converted and sent to the \_Base64In() function. Audio that was text was converted to will play at the select language, pitch, speaking rate, and audio configurations. The \_Base64In function saves the audio internally to Misty which can be played using the misty.PlayAudio() method.

Function \_**IsReady**( , ) – accepts two strings. Compares the current global variables which hold the user’s actual response and the two responses it has reactions to. The similar( , ) function is called twice in which its value is saved within another variable. These two variables percentage values are then compared to 65%. Misty will respond to the specific utterance which similar percentage is greater than 65%.

Translation  
This skill is an applied use of two skills defined before, Sound To Text and Text To Sound. The translation part is the middle of the skill. For understanding of the other parts used, please read the documentation for the other two skills.

The translation part uses only two functions: \_translatedData and translateText.

Our translateText function needs only one parameter, which is a string representing the text that you want to translate. Inside this function, you will need to set up the variable arguments. It is important to change the “source” and “target” based on the languages that you want to translate from and to. Then, we use the SendExternalRequest function to call the Google API, and set our last argument to the event in which the return values should be sent to, in this case \_translatedData.

The \_translatedData function will get the return value from the Google API as its only parameter. Then, it will save to a variable that can be accessed from other functions.

Our JSON file in this skill plays a very important part and have a different set up. Inside the “parameters”, there are three variables called: "APIKEY\_GoogleSTT", "APIKEY\_GoogleTTS", and "APIKEY\_GoogleTranslate” which are followed by a colon and then a string. These strings are the keys that you should get from google. Each key should be entered with the service that you want to access. Explanation on how to get this key is under “Installation and Requirements to work with This Project”.

Tic-tac-toe  
This skill utilizes Misty’s DisplayImage() methods to simulate a tic-tac-toe game live on her display. The skill begins by clearing all possible images then displays a blank nine tile board on her display. Although you may randomly decide which Misty would go first with some minor tweaks, computer2 makes the first move. The GameStart() function starts the game where the computer2(Misty2) makes the first move. This skill simulates the game by randomly selecting a tile that is available depending on which computer’s turn it is. There originally was a plan to implement the minimax algorithm so that our Misty could statistically make to best move during play. However, priority was getting information from the dashboard to that it may communicate to another robot. Global variable playerturn, activegame, and boardstate were going to be posted to our dashboard. Playerturn would equal Player1 or Player2 while activegame equals true or false and boardstate contains an array of nine characters. The idea we wanted to implement was one player making their turn in which after their turn is done, that Misty (or player) would update the dashboard by changing the value of playerturn to the other player and the updated boardstate with their selected tile. We were not able to implement communication through the dashboard so you may simulate tic-tac-toe by running the skill and see a game play until computer1 or computer2 wins or the game ends in a draw. Once the game ends, an animation plays depending on the end result. The game board and all the tiles are cleared off Misty’s screen. You will see that the Reset\_Board() function can be altered to not delete the board itself. This would be useful if you want the game to be able to restart. You may also incorporate Misty’s bump sensor methods so that a live player can interact with the board.

**Functions:**

Function **Reset\_Board**() **–** Deletes all images(board, Xs, Os, game over image). You can implement this or versions of it anywhere within the code. We call this function at the beginning of the code just in the case that the board wasn’t deleted from the screen in a previous run. Otherwise as long as you let the skill run the complete game simulation, the board and all its contents will be cleared off the screen.

misty. DisplayLayerImage() – Built in Misty method that was used to both display the Xs and Os over the blank board, but also to manipulate how and where the image appears on Misty’s display. This method unfortunately only has preset locations on her display you can designate the image to appear. You can set a picture to the top left quadrant, left center, top right, center left, center center, center right, bottom left, bottom center, and bottom right of the display. Due to this, we must manually edit in extra whitespace on certain tiles to get it to fit into the board.

Function **GameStart**() – the primary function starts the game from a blank board state. This function utilizes the HandleTheResults(), DisplayLayerImage, and randomResponse Functions. As long as the state of the game is true(ActiveGame global variable), computer1 and 2 will continue making moves. The HandleTheResults() function switches the global variable ActiveGame to false when the game ends on any condition. The while loop of GameStart() will end and then it calls Reset\_Board to clear Misty’s display so you can see her eyes again.

Function **HandleTheResults**() – Uses the CheckWinState() function to check if the game has ended in a side winning or a draw. It then checks three conditional statements that activate depending on the result of the game. If player1 wins the game, an image displays showing that player1 wins then plays the game over animation. Also sets the ActiveGame to false for all three scenarios.

Function **CheckWinState**() – Compares the contents of the board to all winning combinations after reading in contents of StateOfGame into 9 separate variables. If a winning combination is registered, it also determines which player won. Global variables p1win(Xs) or p2win(Os) will be set to true. The HandleTheResults() function employs this function so it knows if the game ended in a win or lose for player1. The global variable DrawGame will be set to true if there is no winner.

Function **randomResponse**() – Simply decides randomly between two audio files that play on Misty as a reaction to computer2’s move(Misty2).

Function **Computer1Move**(), **Computer2Move**() – Reads in the contents of the board array then randomly selects a tile that hasn’t already been selected previously. If one of the functions picks a tile that has already been selected, it simply generates another random choice until it picks an unoccupied tile. You may develop your own logic inside these functions to dictate why and how Misty makes a move. We simply didn’t have time to implement it because the dashboard functionality wasn’t complete.

Function **UpdateStateOfGame**( , ); - accepts two parameters. One containing the move(the tile selected) and turnof which contains a character ‘X’ or ‘O’). X or O is copied into a temporary variable CurrentState by manually updating all nine tiles no matter which one is selected. The global variable StateOfGame is updated with this function after every move. StateOfGame is copied to the temporary variable CurrentState. CurrentStates contents are read into pick1 – pick9. Depending on the tile selected, an X or O will be copied into one of the pick variables inside the switch case statement. Only the selected tile will be updated while the rest of the contents of the array are copied back into CurrentState. StateOfGame then copies the contents of CurrentState so the global variable’s contents may be utilized elsewhere. StateOfGame is also vital to the CheckWinState function.

**IMPORTANT NOTE ABOUT ARRAYS –** for whatever reason, Misty incorporates commas into the indexes of her arrays. This is important for how our skill runs because instead of an array simply holding 9 characters (0-8 index), it comes out to 17 indexes. So our blank board as you can see starts out blank with underscores: StateOfGame = [ \_ , \_ , \_ , \_ , \_ , \_ , \_ , \_ , \_ ]. This is the only reason why we created the UpdateStateOfGame() function. We were unable to use a for loop to traverse the contents of the board because of the commas. If you doubt this, try running misty.Debug(StateOfGame [0 – 16] in a for loop which will display commas at odd indexes.

[0][2][4][6][8][10][12][14][16] accesses all 9 indexes that contains Xs,Os, and underscores.

Security Camera  
Misty calls a method that starts the video streaming, after that the user opens the website: <http://sdk.mistyrobotics.com/stem/senses.html>. It is possible to see through misty what is happening in the room (within Misty’s camera range). Then we call another method to be able to cancel the video streaming.

For this skill, there is nothing different in the JSON file from what the Misty SDK website generates.   
  
Reaction  
Misty identifies a human, then it tries to do an estimative of which pose the person is, then reacts to that.

The most important part of this skill is the \_human\_pose\_estimation. All the interactions that the Robot will perform is inside this method. Everything else is setting up the robot and allowing it to a lot of math and statistics to perform the pose estimation. It is highly advised to not change anything that is not inside the \_human\_pose\_estimation method.

In this skill, the human body is defined as follow:

NOSE(0), LEFT\_EYE(1), RIGHT\_EYE(2), LEFT\_EAR(3), RIGHT\_EAR(4), LEFT\_SHOULDER(5), RIGHT\_SHOULDER(6), LEFT\_ELBOW(7), RIGHT\_ELBOW(8), LEFT\_WRIST(9), RIGHT\_WRIST(10), LEFT\_HIP(11), RIGHT\_HIP(12), LEFT\_KNEE(13), RIGHT\_KNEE(14), LEFT\_ANKLE(15), and RIGHT\_ANKLE(16).

The only parameter that the \_human\_pose\_estimation takes is a return value from an event. Then, we will store the key points from the human that misty was able to detect inside an array. From there, we try to estimate the pose based on the key points and define an action for misty based on what she saw.

In the JSON file, the only thing that needs to be changed from the generated template in the Misty SDK website is to add an argument called "SkillStorageLifetime" and set its value to "LongTerm". Since we will need to have access to global variables through the code.

Face Recognition  
Misty recognizes faces and then reacts differently to each person it recognizes.

The most important function is the \_FaceRec(). In this function, Misty checks who is that face detected. After that, Misty displays an expression of joy, and depending on the number of times the robot has seen that person, a different image and phrase is displayed by the robot.

In the JSON file, the only thing that needs to be changed from the generated template in the Misty SDK website is to add an argument called "SkillStorageLifetime" and set its value to "LongTerm". Since we will need to have access to global variables through the code.

Rock Paper Scissors  
In this skill, Misty plays rock paper scissors. There are two options: play against human, play against another Misty.

At the moment, the most important function is the \_PlayWithHuman(). In this function, we get a random number. Then the robot displays a countdown before starting the game.

After that, Misty displays its choice randomly. The face is reset to an expression and the game starts all over again.

For this skill, there is nothing different in the JSON file from what the Misty SDK website generates.

Start with IP Address

The only thing that this skill does is show the IP address of the robot on its screen once you tap on the robot’s chin.

We have only 3 functions in this skill. \_Touched, toggleIPLayer, and \_GetDeviceInformation.

The \_Touched function simply checks if the chin of the robot was touched and reacts accordingly.

The toggleIPLayer function is used to configure the display of text in the robot’s screen.

Lastly, the \_GetDeviceInformation is used to actually show the IP in the robot’s screen.

In the JSON file, the only thing that needs to be changed from the generated template in the Misty SDK website is to add a string inside the “StartupRules” called “Startup” right after the “Robot”. Be sure to also add a comma to separate the string. Since we will need to start the skill whenever the robot turns on.

**Installation and Requirements to Work With This Project**

How to get Google API keys

Graphical user interface, text, application, email

Description automatically generated

1. Access <https://cloud.google.com/apis> and select the view API manager button.

Graphical user interface, application, email

Description automatically generated

2. Select tab to the top left portion of the page directly to the left of Google Cloud Platform.

Graphical user interface, text, application

Description automatically generated

3. After you select the project tab you will see all your current projects. You may start new project by clicking the new project option to the top right of select a project.

Graphical user interface, text, application, email

Description automatically generated

4. Input a project name of your choice then click create. You will then be redirected to the project page and most likely receive notification that it was successfully created.

Graphical user interface, application, Word

Description automatically generated

5. Now you will want to search for the API you want to work with. (Above photo)

Graphical user interface, text, application, email

Description automatically generated

Here we selected text-to-speech and speech-to-text APIs.

Graphical user interface, text, application, chat or text message

Description automatically generated

6. Proceed to click enable. You will need to set up a payment plan if you don’t already have a card on file. When you first sign up there is a trial you may use to your advantage. You may look over pricing options and packages, documentation, and support for the selected API.

Graphical user interface, text, application, email

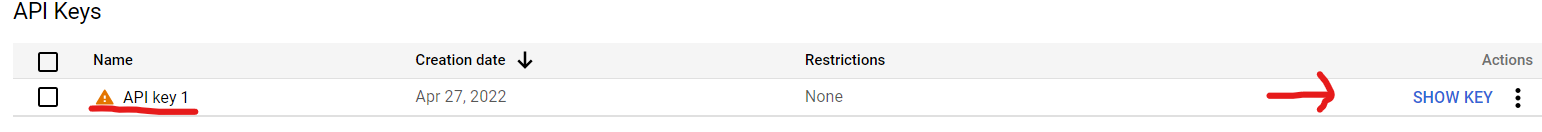
Description automatically generated

7. After you enable the API, you will be sent here. In order to use the API you enabled, you must create an API key that coincides with it. Select the credentials option under APIs and Services.

Graphical user interface, text, application, email

Description automatically generated

8. From the credentials page click + Create Credentials and select the API Key.

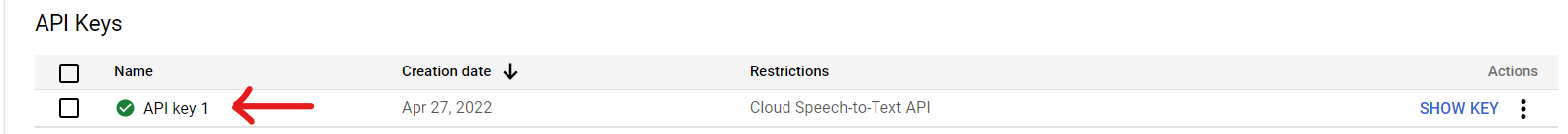


9. After you create the key, you will be sent back to the credentials page where you will see your newly generated API key. Click on the API key 1 because you need to set a restriction.

Graphical user interface, text, application, email

Description automatically generated

10. From here you may set restrictions as you please, but our team simply set a restriction under API restrictions then selected the Speech-To-Text API through the drop down and click save.



After saving the restriction for you API key, you will be sent back to its front page. Here you will notice a green checkmark next to the API key. Your key is now protected by restrictions. You may also copy your key to clipboard by first accessing your key through show key.

You may now place this API key in your JSON to use with you skills. (Check technical manual for guidance on where to place your keys.)

Getting Our Code from GitHub

In order to get access to the source code of this project, please go to the following link:

<https://github.com/HMedeirosDosReis/MistySkills>

A screenshot of a computer

Description automatically generated



Then click the green Code button:

A screenshot of a computer

Description automatically generated



Click on Download ZIP and place it on your preferred directory.

Extract the folder. Then you will be able to see and edit the code.