To explain how the Javascript of my virtual pet works, I categorized the main parts of my code into three parts. The first part includes the stimuli and reactions functions. This includes the `setTimeout()` function and `current` variable. The second part explains the spontaneous hunger or sleep effect, which enables Gudetama to get hungry or sleepy randomly, and includes the `setInterval()`. The third and final part explains the loading effect so the user cannot spam the buttons, using CSS to create a loader.

Stimuli and Behavior

Each of the four button, feed, poke, play, and sleep, have an onclick calling its corresponding function name. The functions contain many if-else statements, all according to the state machine table. In each if-else statement consists of updating the shown image, updating `current` variable, and the `loading()` function.

The `current` variable changes throughout the process, always matching the image. This variable is checked in the if-else statements to decide which behavior to do next.

Below is an example of the `pokeIt()` function.

```
function pokeIt() {
      if(current=='content' || current=='angry') {
            document.getElementById('buttonDiv').style.visibility =
                  "hidden";
            document.getElementById('gifimage').src = poke;
            document.getElementById('state').innerHTML = "Poking...";
            loading(true);
            setTimeout(function myFunction(){loading(false); happyState();},
                  1500);
      } else if( ... ) {
       ...
      }
 ...
}
```

The examples reads: if `current=='content'` or `current=='angry'`, then `setTimeout()` will run `happyState()`. The other code is for the loading function which will be explained later. The function `happyState()` is shown below. As shown, the `current` variable is set to `happy`, the image is updated, and the state on the web page reads `"Happy"` .

```
function happyState() {
      current = 'happy';
      document.getElementById('gifimage').src = happy;
      document.getElementById('state').innerHTML = "Happy";
}
```

I use the DOM method to grab the id of `buttonDiv`, `gifimage`, and `state`. The id `buttonDiv` is the of all the buttons. In this case, I hid all the buttons. The id `gifimage` is the id for <img> tag in the HTML web page. The id `state` is the text next to the image in the HTML web page.

The image is updated by changing the DOM `.src` image property to the variable `happy`, which is a string of the image location. I decided to create variables for all the images so I could change them when I upload them to the tux server (image locations change). There are a total of 18 variables holding the image locations.

There are 9 total functions similar to `happyState()` function above. This way, there is less repetition in the if-else statements.

Gudetama starts at the content state, enabled in the onload event function `init()` . The function `init()` also includes adding the onclick to all buttons and running the `setInterval()` for the `randomNeed()` function.

Spontaneous Hunger and Sleep

Gudetama gets hungry or sleepy (never at the same time) every 20 seconds, unless he is already. A `setInterval()` function is started onload as explained in part 1 and is repeated every 20 seconds nonstop.

```
function randomNeed() {
  if(current=='sleepy'||current=='tired'||current=='hungry'||current=='starved') {
    //Will only run if not hungry or sleepy
  } else {
    if(Math.random() >= 0.5) {
      document.getElementById("buttonDiv").style.visibility = "hidden";
      document.getElementById('gifimage').src = hungry;
      document.getElementById('state').innerHTML = "Random Hunger!...";
      loading(true);
      setTimeout(function myFunction(){loading(false); hungryState();   },
            2000);
    } else {
```

```
      document.getElementById("buttonDiv").style.visibility = "hidden";
      document.getElementById('gifimage').src = sleepy;
      current='sleepy';
      document.getElementById('state').innerHTML = "Random NapTime!...";
      loading(true);
      setTimeout(function myFunction(){loading(false); sleepyState(); },
      2000);
    }
  }
}
```

The `randomNeed()` function above only runs if current equals anything but `sleepy`, `tired`, `hungry`, and `starved`. When conditions met, there is a 50/50 chance Gudetama will be hungry or sleepy.

Loading Effect

To prevent user spamming buttons and to make Gudetama seem to be interacting with the user, I added the loading effect. A loading icon that moves appears, all the buttons disappear, a transition state appears, and the text changes for 2-3 seconds, then the code continues and changes to the next state, the loading icon disappears and buttons appear again.

Each if-else statement contains the code below.

```
document.getElementById("buttonDiv").style.visibility = "hidden";
document.getElementById('gifimage').src = //transition state here;
document.getElementById('state').innerHTML = //transition text here;
loading(true);
setTimeout(function myFunction(){loading(false); //next state function here },
      2000);
```

Loading icon appears, buttons disappear, transition image and text is shown. After 2 seconds, the next state function runs. The `loading()` function justs makes the loading icon appear if set to true, and disappear when set to false. Some CSS code art creates the spinning effect on the icon.