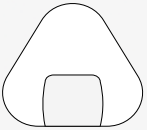
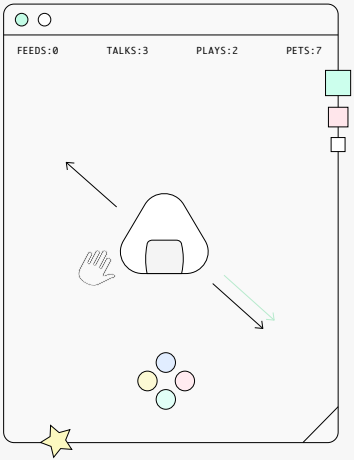
My Friend, Onigiri

This document serves as supporting material for my university applications.





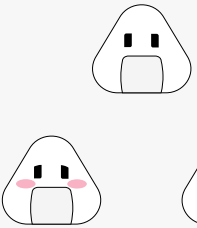
"My Friend, Onigiri" is a browser-based game built with HTML, CSS, and Javascript. It utilises HTML5'S local storage API to save the player's progress throughout the game.

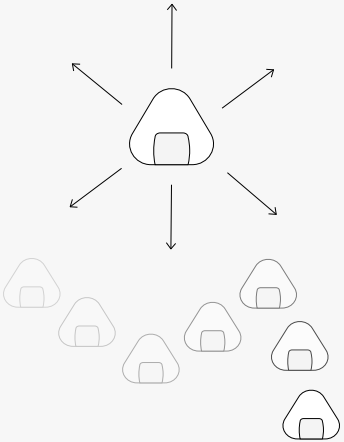
The concept is simple: the player is first presented with a slumbering onigiri. Upon waking the onigiri, the interface pieces together and the game begins. Similar to your run-of-the-mill virtual adoption game, the player is given options to play with, talk to, feed, and pet the onigiri. However, because the onigiri is inevitably a food, the player is given one more option: to eat it.

The vast majority of approximately ***200 beta players confessed to eating the onigiri*** during their first run, whether out of curiosity or simply for amusement. Interestingly enough, the players who chose to eat their onigiri felt guilty afterwards, and restarted the game. During their second run, those players collectively spent more time earning rewards and building their relationship with the onigiri instead. There is no correct or incorrect way to play the game, but it's reasonable to conclude that most players are tempted to eat the onigiri at least once. (The game was released for testing on Tumblr).

Gameplay consists of player ***interactions*** with the onigiri. While the feed, play, talk, and eat interactions are represented by clickable buttons, the petting interaction requires the player to hold the mouse down and run the cursor over the onigiri. On mouse down, the cursor turns into a hand, creating the pretense that the player truly is petting the onigiri. This results in a positive reaction from the onigiri, which becomes apparent to the player in its facial expressions.

Throughout the game, the onigiri shows a number of expressions, ranging from sleepy to bashful to ravenous. ***The game's image library contains over 50 onigiri expressions***, each triggered by an event. The first of these events is loading of the window. The game will first check the player's local time; if the time returned is between 8PM to 7AM, the onigiri will be asleep and will require a tap from the player to wake.





The fun of the game is in the ***animations***. Two styles of animations were used to bring the onigiri to life: GIF animations in which the onigiri continuously bounces up and down, and JQuery animations triggered by events. Interacting with the onigiri in any way will trigger an animation. For example, playing with the onigiri will prompt it to jump around in glee. Speaking to the onigiri will trigger a range of animations, from nervous shuffling to retrieving an object from the sidelines and carrying it onto the game screen. However, there is one animation that executes itself continously throughout the game. From the moment the game begins, the onigiri will begin to dance across the screen until the player interacts with it, causing it to move in another direction.

```
function dance(){
  setInterval(function(){
    var x = Math.random() * 100;
    var y = Math.random() * 100;
    $("#oniCage").animate({
      left: x + '%',
      top: y + '%'
    }, 4000);
  }, 4000);
}
```

Animations were also used to create mini movies, known in the game as ***rewards***. Each mini movie ends by either leaving a new interactive object on the screen or by presenting the player with a new privilege, like the ability to give their onigiri a name. The rewards system works by incrementing a global variable each time the player interacts with their onigiri. Once the player has used a particular interaction enough times, a new reward will be triggered.

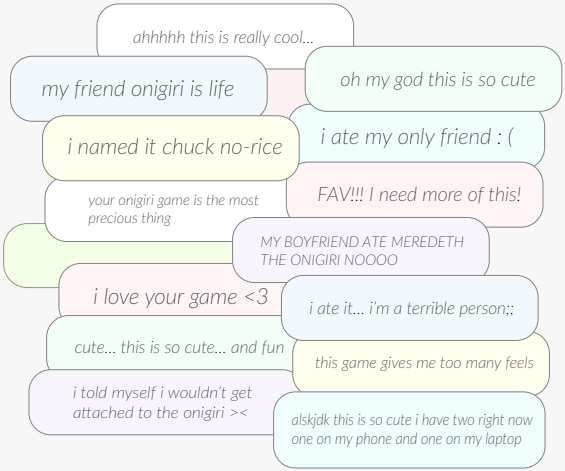
```
playButton.onclick = function(){
  { ... animation block ... }
  ++playCount;
  if ( playCount == 25 ) {
    { ... reward block ... }
  } else if ( playCount == 50 ) {
    { ... reward2 block ... }
  }
};
```

Depending on the interaction chosen by the player, the onigiri will either perform the corresponding animation in its current position on the screen, or centre itself again if the animation requires a larger amount of space than available. In order to prevent glitching, a large, invisible divider is set to `display:block` during each animation sequence, effectively preventing the player from triggering any more animations until the current animation is complete.

Local storage and ***player input*** was heavily utilised to personalise the game experience to each individual player. One example is in the beginning of a new game when a player is prompted to enter their name. A local storage item saves the name for their next visit, and a switch case block determines which colour the game's backdrop will be using the first letter of the player's name. The information the player is prompted to enter both at the start of the game and throughout the game re-emerges in conversations and mini movies. Another instance of this is the use of the player's age. If a player whose age is less than nineteen opens the game on a weekday after three o'clock in the afternoon, instead of using a standard afternoon greeting, their onigiri will welcome them home from school.

```
startNewGame.onclick = function(){
  pName = enterName.value;
  localStorage.setItem("nm", pName);
  switch (pName[0]) {
    case "A":
    case "B":
    BG.style.background="#FFD1DD";
      break;
    case "C":
    case "D":
    BG.style.background="#D3C4FF";
      break;
    ...
  };
};
```

All in all, the game was simple in design and implementation. Being that it was the first game I'd ever designed and programmed, it is as much a milestone as it was an adventure. To conclude this brief look at "My Friend, Onigiri", here are a few closing comments from players of the game.



Thank you for reading!