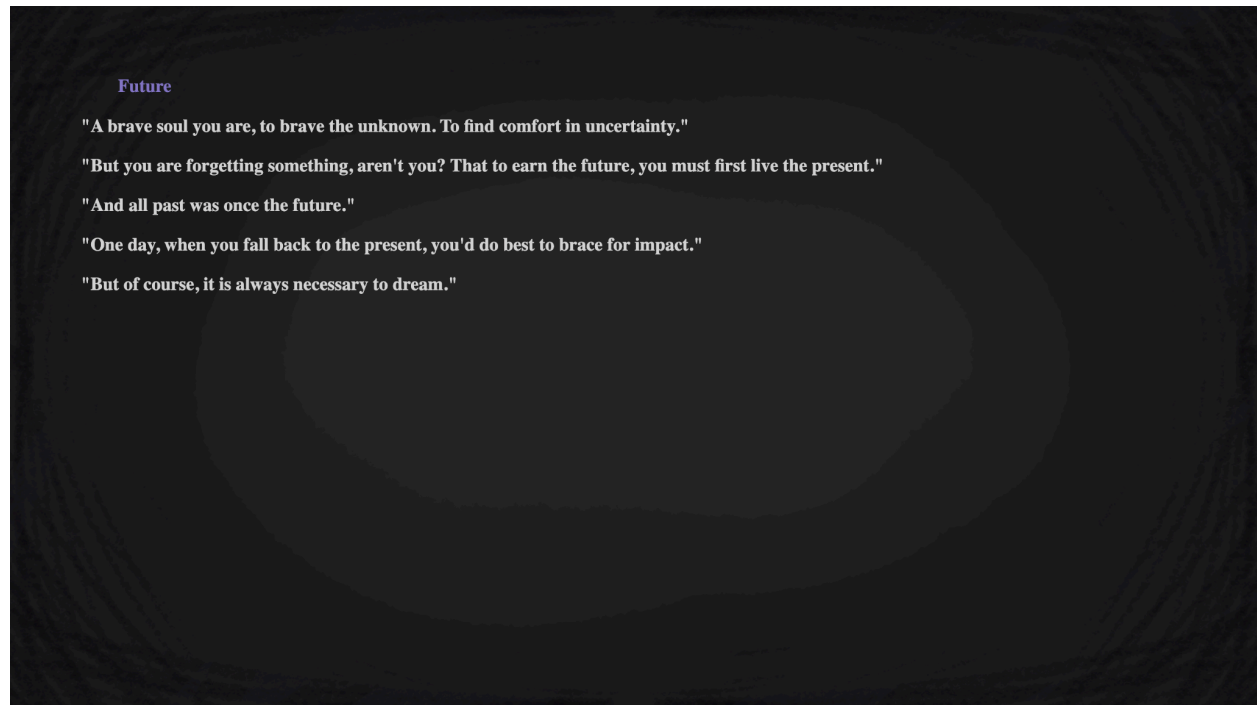


the self in the mirror the mirror in the self

By Vanessa Ngai



Caption: Some of the dialogue from the game. The player's choice is in purple, and the narrator's lines are in white. The game is advanced by clicking on the newest text.

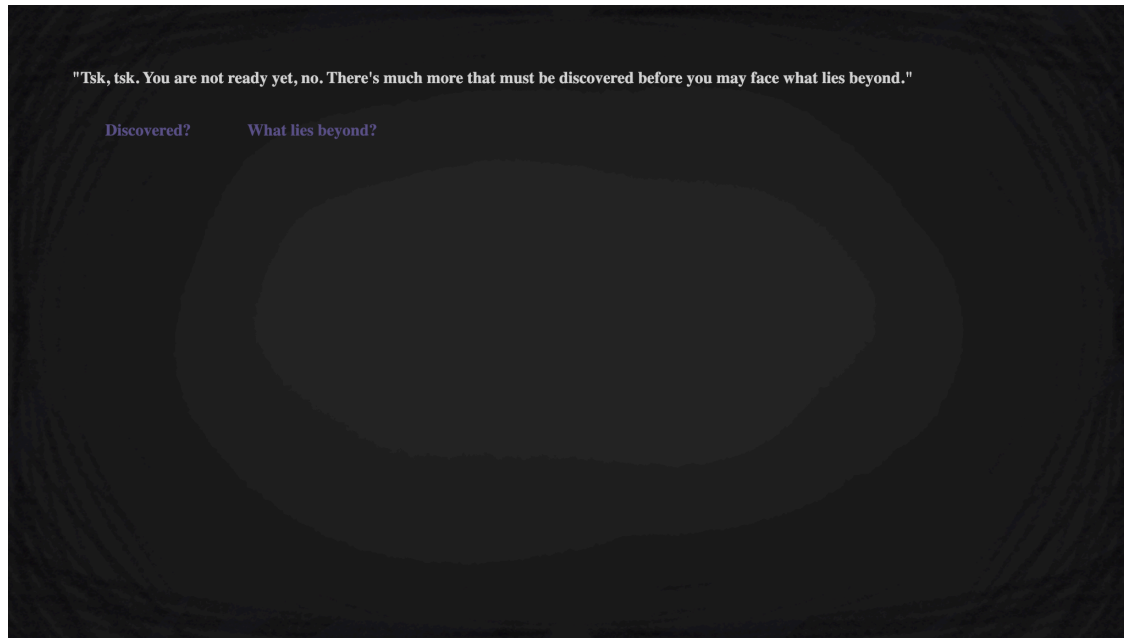
Showcase / Description of Finished Piece



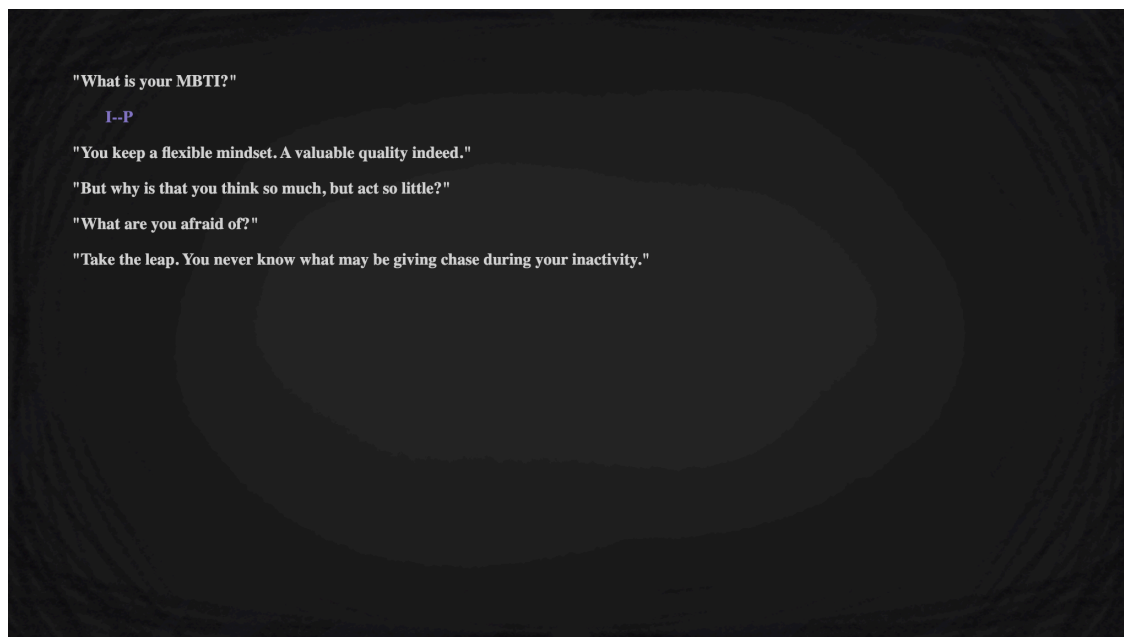
The game immediately opens to a mirror prompting the player to “Come closer.” Clicking on the mirror activates the device’s webcam, reflecting the player on the mirror surface. I wanted to bring the player into the experience more: by seeing themselves, there is a feeling of reality and being watched. The dark, simple design of the page creates a mysterious atmosphere and pushes the player to the forefront.



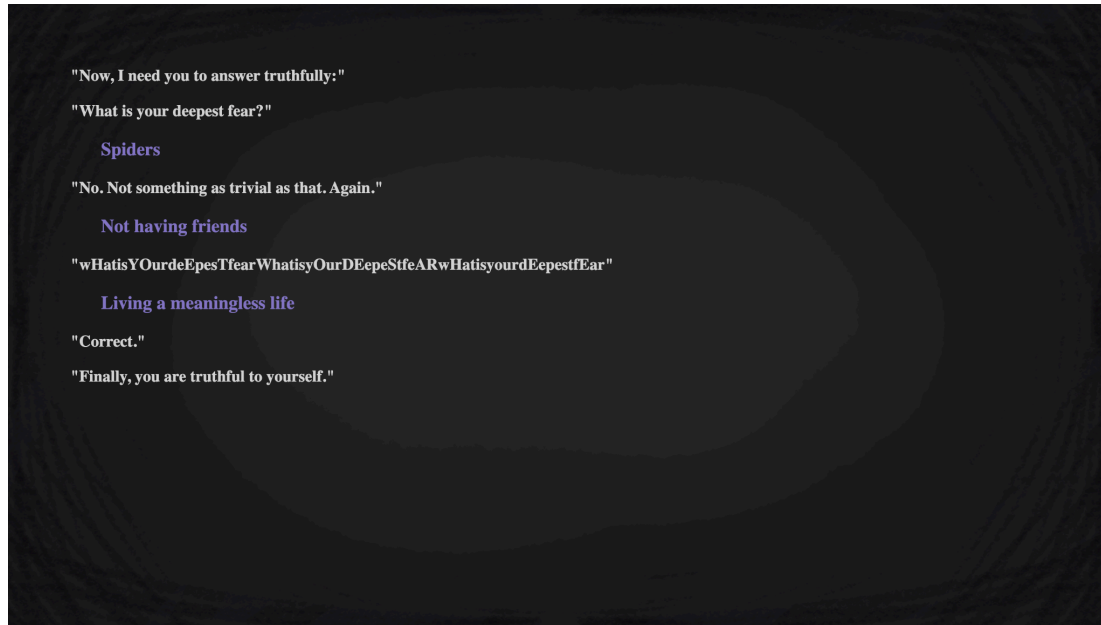
After the mirror is an odd door. The player will be told that they cannot progress past this point yet– this establishes the “goal” of the game early on.



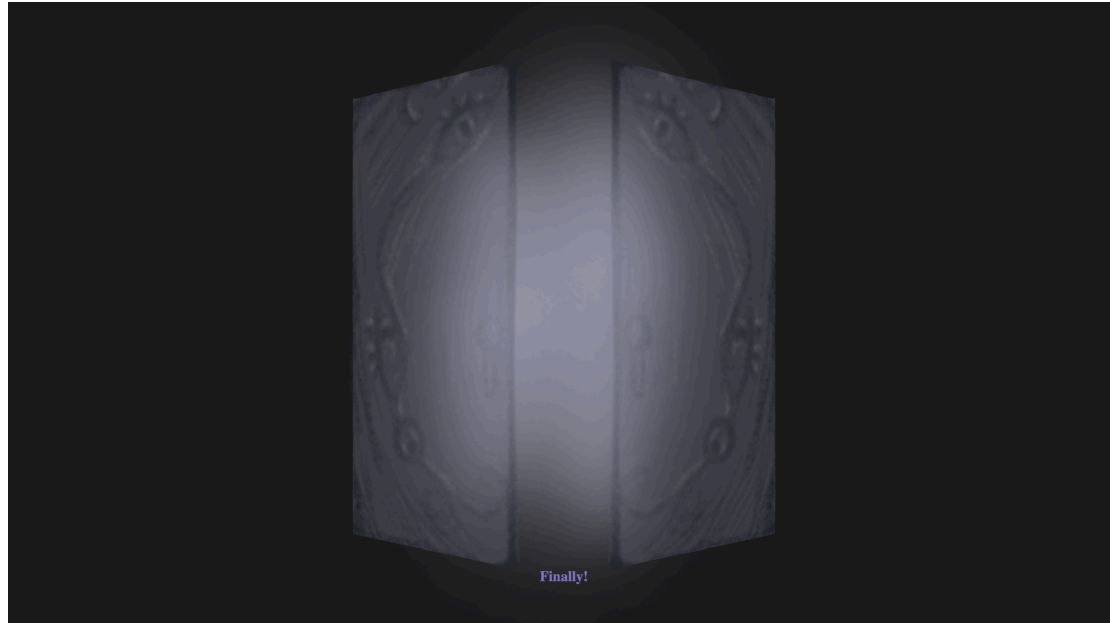
The player will then make their first choice of the game, introducing the main mechanics of the experience. Clicking on the most recent line of text causes the next line to appear, and purple text indicates a decision that the player can make.



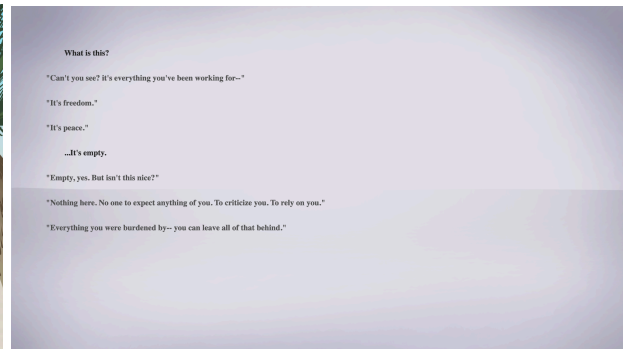
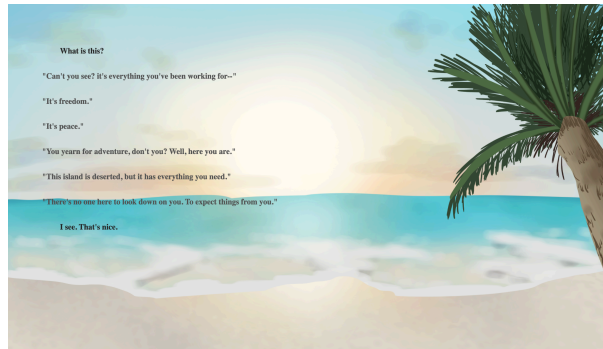
While the first few questions are typical of those found in personality quiz games, the narrator gets increasingly critical of the player and shifts into a persona of pseudo-omniscience.



In one of the most intrusive, yet reflective moments of the experience, the player is asked to type in their deepest fear. This introduces a new gameplay element from the established routine that unsettles the player, making the subsequent unexpected responses even more impactful.



After the question about fear, the player has finally completed the question/answer segment of the game, and they are brought back to the door.



Behind the door leads the player to one of three endings, depending on their performance. After looking around for a bit, the narrator says a few closing words to the player, and they are sent back to the opening screen.

[Link to Demo Video](#)

This piece is my idea of self exploration. The personality-quiz style gameplay is a core component of my belief that everyone spends their lives trying to understand themselves. People generally understand that personality tests are not reflective of themselves—after all, they are hard coded and made by a stranger who will never know them personally. Despite that, many still find themselves spending time on these things. Perhaps we are willing to take the chance that we will be able to understand something new, no matter how trivial or how unlikely. In my piece, I try to make the experience worthwhile in this regard by pushing the player to reflect on themselves.

Process

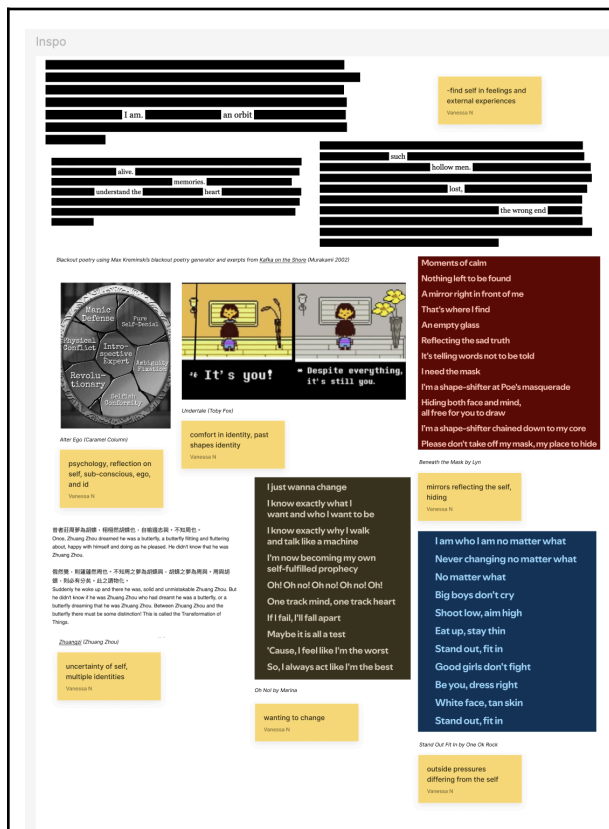


Figure 1: I started with collecting sources of inspiration relating to the theme: sense of self. I found pieces that related to self exploration, identity, or even self-critique.

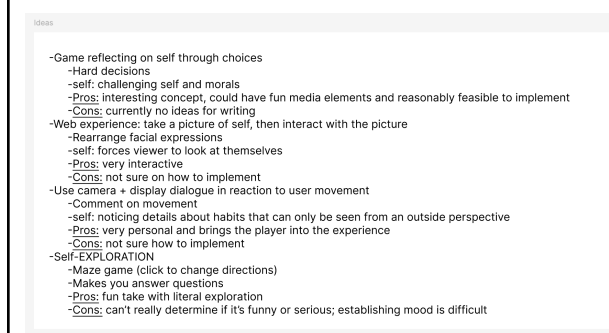
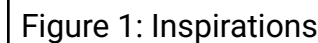


Figure 2: After gathering inspiration and seeing a common theme in each unique piece, I started to think of ideas for my piece.

Ideation/Design Process (pg 1 of 3):
View original document [here](#).

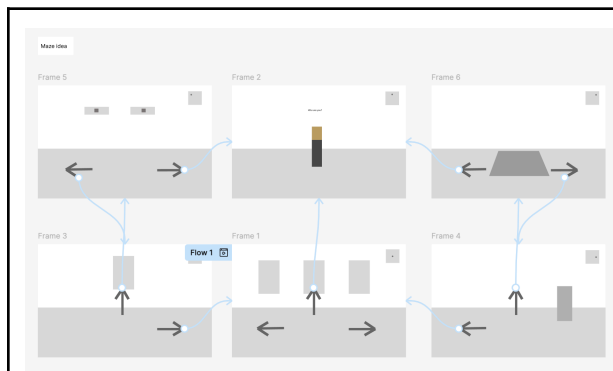


Figure 3: Low-fidelity wireframe (maze exploration)

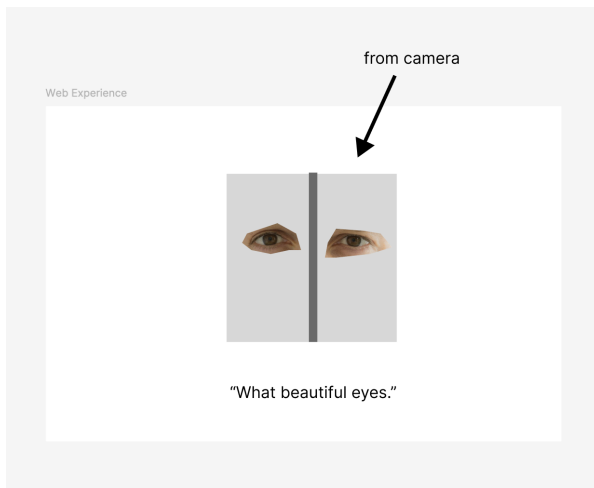


Figure 4: Key visual (web experience)

Figure 3: I made a low-fidelity flow of the gameplay for the most concrete idea I had, the maze exploration. The player would use arrows to move through the maze and be faced with interesting visuals, which self-reflection questions intermittently sprinkled at key stages. The idea of asking questions was kept in the final product.

Figure 4: I created a sample of the key visual for a web experience that would implement part of the viewer's image into the game, forcing them to look at themselves and think. I thought the concept was very interesting, and this was the original plan that I chose.

Ideation/Design Process (pg 2 of 3):
View original document [here](#).

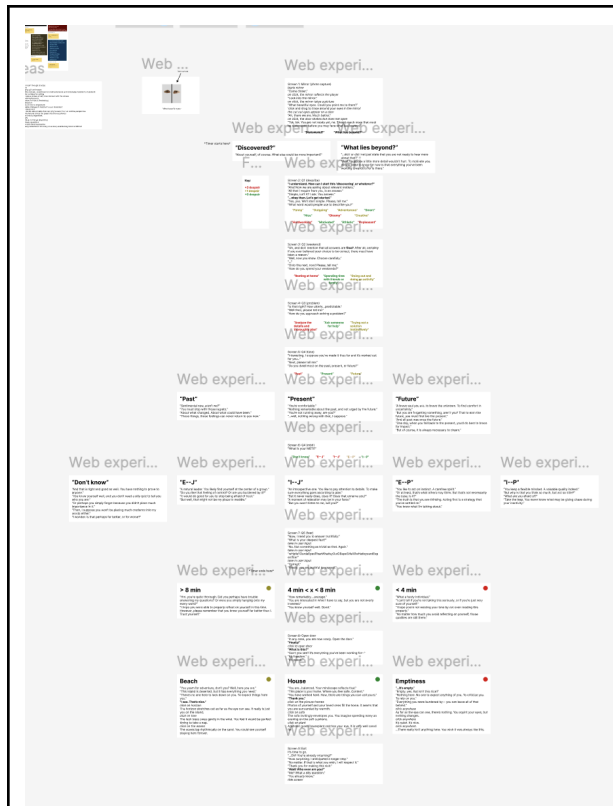


Figure 6: Script

Figure 6: After I decided on my idea, which was a combination of both the prototypes (quizzes and forced self reflection), I wrote out a script with branching paths. I wanted each user experience to be unique. I also wanted to catch the viewer off guard with their expectations. For example, the first few questions, which seemed like they would reveal details about themselves, were actually hard coded with identical responses. However, how long they spent playing the game was a contributing factor to their ending. I also wanted the narrator to play a balance between being an interesting plot-pushing character, yet not the main focus. The narrator starts off as very mysterious, but surprisingly snarky. I wanted it to be balanced between seeming human, yet also all-powerful. As the game progresses, the narrator seemingly gains more control. It subtly tells the player about themselves, but what could a fixed line of letters actually know about a person? Its inexplicable confidence and authority should make the viewer slightly uncomfortable, but this should also remind them about the significance they are placing into the advice from this game and any other outside spectator.

There are 3 possible endings depending on the player's choices, each of which reflect what a person might think of as "utopia".

Ideation/Design Process (pg 3 of 3):
View original document [here](#).

View original document [here](#).

```

≡ requirements.txt
1  blinker==1.8.2
2  click==8.1.8
3  flask==3.0.3
4  importlib-metadata==8.5.0
5  itsdangerous==2.2.0
6  jinja2==3.1.6
7  MarkupSafe==2.1.5
8  numpy==1.24.4
9  opencv-python==4.13.0.92
10 pillow==10.4.0
11 werkzeug==3.0.6
12 zipp==3.20.2
13

```

Figure 7: requirements.txt

```

app.py > ...
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def mirror():
7      return render_template("mirror.html")
8
9  if __name__ == "__main__":
10     app.run(debug=True)

```

```

templates > mirror.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>The Mirror</title>
5      <link rel="stylesheet" href="/static/styles.css">
6  </head>
7  <body>
8      <div class="landing">
9          <h1 style="color: white; padding: 5vh;">the self in the mirror the mirror in the self</h1>
10         <div class="mirror-container">
11             
12             <div class="mirror" id="mirror">
13                 <video id="camera" autoplay playsinline</video>
14             </div>
15         </div>
16         <h2 class="mirror-caption" id="caption1">Come closer.</h2>
17         <h2 class="mirror-caption" id="caption2">Look into the mirror.</h2>
18     </div>

```

Figures 8 & 9: Set up first HTML file and app.py

Figure 7: The first thing I did to set up my project was create a requirements file and install dependencies. I knew that I was going to be using multiple pages and would need to use Flask and host it on a local server. I used

None

```

pip freeze >
requirements.txt

```

to save my dependencies in a requirements file for easier distribution and setup.

Figures 8 & 9: My next step was to start the very first file, which would be the [app.py](#) file that ran the program. I then created my first HTML to start building the actual frontend. I purposely wanted it to be very simple even if that made it vague, as to encourage exploration. As such, I just set up the simple text and styled it in the separate CSS.

Prototyping/Building Process (pg 1 of 7):
View original document [here](#).

```

21 + // Request camera access
22 + const mirror = document.getElementById("mirror");
23 + const video = document.getElementById("camera");
24 + const capture = document.getElementById("capture");
25 + const capture2 = document.getElementById("capture2");
26 +
27 + let stream = null;
28 + let cameraActive = false;
29 + let photoTaken = false;
30 +
31 + // First click = activate camera
32 + mirror.addEventListener("click", async () => {
33 +   if (navigator.mediaDevices.getUserMedia) {
34 +     // If (getUserMedia) returns
35 +     stream = await navigator.mediaDevices.getUserMedia({ video: true });
36 +     video.srcObject = stream;
37 +     video.style.display = "block";
38 +     capture.style.display = "none";
39 +     capture2.style.display = "block";
40 +     cameraActive = true;
41 +   } else {
42 +     // If (getUserMedia) fails
43 +     photoTaken = true;
44 +   }
45 + }
46 + );
47 +
48 + function capturePhoto() {
49 +   // Create canvas to hold the captured frame
50 +   const canvas = document.createElement("canvas");
51 +   canvas.width = video.videoWidth;
52 +   canvas.height = video.videoHeight;
53 +
54 +   const ctx = canvas.getContext("2d");
55 +   ctx.drawImage(video, 0, 0, ctx.canvas.width, ctx.canvas.height);
56 +
57 +   // Freeze the image in the mirror
58 +   // Replace the video element with the canvas
59 +   video.style.display = "none";
60 +   canvas.id = "frozenPhoto";
61 +   canvas.style.width = "700px";
62 +   canvas.style.height = "700px";
63 +   canvas.style.position = "absolute";
64 +   canvas.style.top = "0px";
65 +   canvas.style.left = "0px";
66 +
67 +   const mirror = document.getElementById("mirror");
68 +   mirror.appendChild(canvas);
69 +
70 +   // Optionally, stop the camera stream to save resources
71 +   if (stream) {
72 +     stream.getTracks().forEach(track => track.stop());
73 +   }
74 +
75 +   // Convert canvas to base64 to send to backend
76 +   const imageData = canvas.toDataURL("image/png");
77 +
78 +   fetch("api/eyes", {
79 +     method: "POST",
80 +     headers: { "Content-Type": "application/json" },
81 +     body: JSON.stringify({ image: imageData })
82 +   });
83 + }

```

```

27 .mirror {
28   width: 300px;
29   height: 700px;
30   border-radius: 80%;
31   overflow: hidden;
32   position: relative;
33   background-color: #gray;
34   cursor: default;
35 }
36
37 .mirror:hover {
38   cursor: pointer;
39 }
40
41 .frame {
42   position: absolute;
43   width: 170%;
44   height: 140%;
45   top: -150px;
46   left: -110px;
47   pointer-events: none;
48   z-index: 2;
49 }
50
51 /* Camera feed fills mirror */
52 .mirror video {
53   width: 100%;
54   height: 100%;
55   object-fit: cover;
56   animation-name: fadeIn;
57   animation-duration: 5000ms;
58   animation-delay: 1000ms;
59 }

```

Figures 10 & 11: Implement mirror



Figure 11: Attempt at masking

Figures 10 & 11: For the mirror, I created a video object to display the device's camera feed, then used CSS to shape the video container into an oval and remove any distortion. I took a screenshot as a size reference to draw the frame for the mirror. I imported this into the project, and used CSS to align it with the video feed.

My initial plan was for the mirror to capture the player, then use that photo throughout the game. I used ChatGPT to create code to take a photo on click, then disable the camera after the photo was taken. [Video of the mirror with the camera capture here.](#)

Figure 12: After taking the photo, the player would outline their eyes, which would then be transplanted onto a door in the next scene. I used ChatGPT to generate code to enable the player to trace their eyes. By clicking and dragging, the player could draw on the photo, and on release, the end of the stroke would automatically connect to the beginning to create a closed shape. The user would make exactly two strokes before the segment ended. Unfortunately, I was not able to apply masking to the shapes to cut out just the eyes from the photo, so I ultimately decided not to keep this element in the final product.

Prototyping/Building Process (pg 2 of 7):
View original code [here](#).

```

1 <!--DOCTYPE html--> <html> <body> <script> <!-- door_closed.addEventListener("click") callback
2 <!--html-->
3 <!--head-->
4 <!--title--The Mirror-->
5 <!--link rel="stylesheet" href="static/styles.css"-->
6 <!--/head-->
7 <!--body-->
8 <div class="main">
9 
10 <h3 class="s1_text" id="s1_text1" style="display: none;"> "Tsk, tsk. You are not ready yet, no.
11 </div>
12
13 <script>
14
15 const door_closed = document.getElementById("door_closed");
16 const s1_text1 = document.getElementById("s1_text1");
17
18 let first_click = false
19
20 door_closed.addEventListener("click", async () => {
21   if (!first_click){
22     s1_text1.style.display = "block";
23     first_click = true;
24   } else {
25     window.location.href="dialogue.html";
26   }
27 });
28
29 function add_shake() {
30   door_closed.classList.add('shake');
31 }
32 </script>

```

```

271 @keyframes shake {
272   0%, 100% {transform: translateX(0%);}
273   20% {transform: translateX(5%);}
274   40% {transform: translateX(-5%);}
275   60% {transform: translateX(5%);}
276   80% {transform: translateX(-5%);}
277 }

```

```

30 // First click - activate camera
31 mirror.addEventListener("click", async () => {
32   if (!cameraActive) {
33     stream = await navigator.mediaDevices.getUserMedia({ video: true });
34     video.srcObject = stream;
35     video.style.display = "block";
36     caption1.style.display = "none";
37     caption2.style.display = "block";
38     cameraActive = true;
39   } else {
40     // Camera is already active - go to next scene
41     window.location.href = "door.html";
42   }
43 });

```

Figures 13 & 14: Connecting the first scene to the landing page

```

7 <body>
8 <!--Id Nomenclature:
9 Text block: tx_s[scene number]
10 Individual text: t[scene number]_t[response block]_tx[response order]
11 Response block: rs_s[scene number]
12 Response choice: s[scene number]_r[response block].[response order]
13 Response block (selected): rs_sl_s[scene number]
14 Response selection: s[scene number]_rs[response block].[response choice]-->
15 <div class="main-dialogue">
16 <div class="text" id="tx_s1">
17 <h3 class="tx" id="s1_t1" style="cursor: default;">"Tsk, tsk. You are not ready
18 <div class="response" id="rs_s1.1">
19 <h3 class="rs" id="s1_r1.1">Discovered?</h3>
20 <h3 class="rs" id="s1_r1.2">What lies beyond?</h3>
21 </div>
22 <div class="response-select" id="rs_sl_s1" style="display: none;">
23 <h3 class="rs-sl" id="s1_sl.1" style="display: none;">Discovered?</h3>
24 <h3 class="rs-sl" id="s1_sl.2" style="display: none;">What lies beyond?</h3>
25 </div>
26 <h3 class="tx" id="s1_t2_r1.1" style="display: none;">"About yourself, of course
27 <h3 class="tx" id="s1_t2_r2.1" style="display: none;">"...did i or did i not <i
28 <h3 class="tx" id="s1_t2_r2.2" style="display: none;">"Well, I suppose a little
29 <h3 class="tx" id="s1_t2_r2.3" style="display: none;">"All you need to know for
30 <div class="response" id="rs_s1.2" style="display: none;">
31 <h3 class="rs" id="s1_r2" style="display: none;">I understand. How can I st
32 </div>
33 </div>

```

Figure 15: Setting up the text-based gameplay

Figures 13 & 14: I started preparing the next scene in a new HTML file, as I thought it would be easier for the entire file to be swapped out rather than constantly refreshing the components of the same HTML file. I drew the door and added JS with a conditional to trigger the dialogue on click. I also used CSS to create an animation for more dynamism.

After finishing the page, I went back to mirror.html to connect it to the new door.html. [Video of the implemented page switch here.](#)

This is the strategy I used for all instances in which there was a change of setting: I would make a new HTML file, completely finish that segment of the game, and then go back and link it to the previous page.

Figure 15: My core strategy for the dialogue portion was that clicking on one line of text would cause the next to appear. To set this up in HTML, I started with creating a wrapper for each scene so I could easily “reset” the page by hiding all the text and unhiding the new scene. The very first scene would be visible, but every other scene in the game had “display: none” applied to it. I then added every line of text from the script into the file. To distinguish between scenes, speaker, and individual lines, I developed a nomenclature so that each line had a unique id. I also separated the narrator and the choices with different classes to apply different stylings to them.

```

147 + //Scene 2 paths
148 + //text
149 + const tx_s2 = document.getElementById("tx_s2")
150 + const s2_t11 = document.getElementById("s2_t1.1")
151 + const s2_t12 = document.getElementById("s2_t1.2")
152 + const s2_t13 = document.getElementById("s2_t1.3")
153 +
154 + //response
155 + const rs_s2 = document.getElementById("rs_s2")
156 +
157 + r_s1_2.addEventListener("click", async () => {
158 +   tx_s1.style.display = "none";
159 +   tx_s2.style.display = "block";
160 +   r_s1_2.style.cursor = "default";
161 + });
162 + s2_t11.addEventListener("click", async () => {
163 +   s2_t12.style.display = "block";
164 +   s2_t11.style.cursor = "default";
165 + });
166 + s2_t12.addEventListener("click", async () => {
167 +   s2_t13.style.display = "block";
168 +   s2_t12.style.cursor = "default";
169 + });
170 + s2_t13.addEventListener("click", async () => {
171 +   rs_s2.style.display = "block";
172 +   s2_t13.style.cursor = "default";
173 + });
174 +

```

Figure 16: Connecting the text together

```

218 //timer
219 //Reference for time counter: https://tinloaf.com/blog/how-to-build-a-stopwatch-with-html-css-js-react-part-2
220 let elapsedTime = 0;
221 let isRunning = false;
222
223 function startStop() {
224   if (isRunning) {
225     elapsedTime = Date.now() - startTime;
226     isRunning = false;
227     console.log(elapsedTime)
228   } else {
229     startTime = Date.now();
230     isRunning = true;
231     console.log('time started')
232   }
233 }
234
235 //Response 1
236 s1_r1_1.addEventListener("click", async () => {
237   r_s1_1.style.display = "none";
238   rs_s1_1.style.display = "block";
239   s1_s1_1.style.display = "block";
240   s1_t2_11.style.display = "block";
241   startStop();
242 });

```

Figure 17: Adding timer

Figure 16: To make the text interactable, every line and container was made into a JS variable and given an event listener to react to clicks. When a line was clicked it would

- Reveal the next line
- Change the cursor back to default so as not to look “clickable” anymore

For choices, it would also hide the choice text and switch to a different class with slightly different styling, so it did not look “selectable”. [Video of what the interaction looks like](#) (in screen recordings, the cursor does not change).

This was the core principle for every single text in the game; when a scene would end, I would hide the entire text container and unhide the new one. For sections with multiple choices, each choice was linked with a different set of dialogue options.

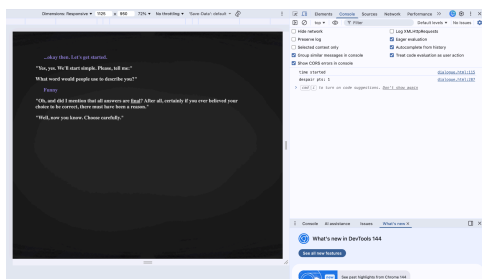
Figure 17: A secret mechanism I wanted to implement was having how long the player took impact their ending. Using resources from the internet, I developed a stopwatch function that started as soon as the first choice was made. To ensure that it was being triggered, I used console logs.

Prototyping/Building Process (pg 4 of 7):
View original code [here](#).

```

329    ---//despair counter
330    ---let despair_pts = 0;
331
332    rs_s2.addEventListener("click", async () => {
333      tx_s2.style.display = "none";
334      tx_s3.style.display = "block";
335      rs_s2.style.cursor = "default";
336    });
337
338    s3_t11.addEventListener("click", async () => {
339      s3_t12.style.display = "block";
340      s3_t11.style.cursor = "default";
341    });
342
343    s3_t12.addEventListener("click", async () => {
344      rs_s31.style.display = "block";
345      s3_t12.style.cursor = "default";
346    });
347
348    ---//Response 1
349    s3_r1_1.addEventListener("click", async () => {
350      rs_s31.style.display = "none";
351      rs_sl_s3.style.display = "block";
352      s3_rs11.style.display = "block";
353      s3_t21.style.display = "block";
354      despair_pts +=1;
355    });

```



Figures 18 & 19: Point system implementation

```

173 <div class="text" id="tx_s0" style="display: none;"
174 <div class="tx" id="s0_t11" style="display: none;">Now, I need you to answer truthfully!</div>
175 <div class="tx" id="s0_t12" style="display: none;">What is your deepest fear?</div>
176 <blockquote class="input" id="input1" contenteditable="true" style="font-style: italic; display: none;" type here...
177 <div class="tx" id="s0_t21" style="display: none;">No. Not something as trivial as that. Again.</div>
178 <blockquote class="input" id="input2" contenteditable="true" style="font-style: italic; display: none;" type here...
179 <div class="tx" id="s0_t31" style="display: none; word-break: break-all;">What is your deepest fear?</div>
180 <blockquote class="input" id="input3" contenteditable="true" style="font-style: italic; display: none;" type here...
181 <div class="tx" id="s0_t41" style="display: none;">Correct.</div>
182 <div class="tx" id="s0_t42" style="display: none;">Finally, you are truthful to yourself.</div>
183 </div>
184
185 ---//Response 1
186 //Reference for "Enter" detector: https://codepen.io/mauricio/pen/now-to-detect-enter-key-in-javascript?&editors=0110
187 input1.addEventListener('keydown', function(event) {
188   if (event.code === "Enter") {
189     input1.contentEditable = "false";
190     input1.style.fontSize = "normal";
191     s0_t21.style.display = "block";
192   }
193 });
194
195 s0_t21.addEventListener('click', async () => {
196   input2.style.display = "block";
197   s0_t21.style.cursor = "default";
198 });
199
200 ---//Response 2
201 input2.addEventListener('keydown', function(event) {
202   if (event.code === "Enter") {
203     input2.contentEditable = "false";
204     input2.style.fontSize = "normal";
205     s0_t31.style.display = "block";
206   }
207 });
208
209 s0_t31.addEventListener('click', async () => {
210   input3.style.display = "block";
211   s0_t31.style.cursor = "default";
212 });

```

Figures 20 & 21: Typing segment implementation

Figures 18 & 19: To determine the ending, I attributed point values of 0, 1, or 3 to each choice the player made, hereby called “despair points”. I created a variable to track the player’s accumulated points, and in each option, I simply added the corresponding amount of points to the variable. Again, to make sure it was working correctly, I added a console log during testing.

Figures 20 & 21: For the typing gameplay in scene 9, I used a blockquote with contenteditable=“True” for the input. Using resources from the Internet, I was able to program the blockquote to stop being editable once the player hit “Enter”. I also changed the font from italics to normal as an indicator that the answer was finalized. [Video of the implemented typing gameplay here.](#)

Prototyping/Building Process (pg 5 of 7):
View original code [here](#).

```

947 s9_t42.addEventListener("click", async () => {
948   startStop();
949
950   tx_s9.style.display = "none";
951   tx_s10.style.display = "block";
952
953   if (elapsedTime < 120000) {
954     despair_pts += 3;
955     s10_t1_r31.style.display = "block";
956   } else if (elapsedTime < 240000) {
957     despair_pts += 0;
958     s10_t1_r21.style.display = "block";
959   } else {
960     despair_pts += 1;
961     s10_t1_r11.style.display = "block";
962   }
963   console.log(`despair pts: ${despair_pts}`)
964   localStorage.setItem("despair_pts", despair_pts);
965 });

```

Figure 22: Ending the timer

```

17 const door_closed = document.getElementById("door_closed");
18 const door_open = document.getElementById("door_open");
19 const s11_text1 = document.getElementById("s11_text1");
20 const s11_r1 = document.getElementById("s11_r1");
21 const despair_pts = localStorage.getItem("despair_pts");
22 console.log(despair_pts);
23
24 s11_text1.addEventListener("click", async () => {
25   s11_text1.style.display = "none";
26   s11_r1.style.display = "block";
27 });
28
29 s11_r1.addEventListener("click", async () => {
30   door_closed.style.display = "none";
31   door_open.style.display = "block";
32   s11_r1.style.pointerEvents = "none";
33 });
34
35 door_open.addEventListener("click", async () => {
36   if (despair_pts < 6) {
37     window.location.href = "empty.html";
38   } else if (despair_pts < 12) {
39     window.location.href = "beach.html";
40   } else {
41     window.location.href = "room.html";
42   }
43 });

```

Figure 23: Triggering endings

```

0 <div class="wrapper">
1   <div class="bg-wrapper">
2     
3     
4     
5     
6   </div>
7   <div class="final_text" id="tx_s12_1" style="display: none;">
8     <div class="text" id="tx_s12_1_1">What is this?</div>
9     <div class="text" id="tx_s12_1_2" style="display: none;">Can't you see? It's everything you've been working for.</div>
10    <div class="text" id="tx_s12_1_3" style="display: none;">It's freedom.</div>
11    <div class="text" id="tx_s12_1_4" style="display: none;">It's peace.</div>
12    <div class="text" id="tx_s12_1_5" style="display: none;">You are...balanced. Your landscape reflects that.</div>
13    <div class="text" id="tx_s12_1_6" style="display: none;">This place is your home. Where you feel safe. Content.</div>
14    <div class="text" id="tx_s12_1_7" style="display: none;">You have worked hard. Now, there are things you can call yours.</div>
15    <div class="text" id="tx_s12_1_8" style="display: none;">Thank you.</div>
16  </div>
17   <div class="final_text" id="tx_s12_2" style="display: none;">
18     <div class="text" id="tx_s12_2_1" style="display: none;">The sofa invitingly envelopes you. You imagine spending many an evening on the soft cushions.</div>
19     <div class="text" id="tx_s12_2_2" style="display: none;">
20       <div class="final_text" id="tx_s12_3" style="display: none;">
21         <div class="text" id="tx_s12_3_1" style="display: none;">Photos of yourself and your loved ones fill the home. It seems that you are surrounded by warmth.</div>
22         <div class="final_text" id="tx_s12_4" style="display: none;">
23           <div class="text" id="tx_s12_4_1" style="display: none;">A vibrant green houseplant catches your eye. It is very well cared for.</div>
24           <div>

```

Figure 24: Setting up ending gameplay

Figure 22: At the end of scene 9, I stop the timer and use conditionals to display different responses depending on the time elapsed. I use the console log to check that this is working properly. [Video of the implementation and console log of the timer here](#). (Note: the time boundaries for each text branch has been changed since the video). I also save the player's accumulated points in local storage so that it will be accessible by other files when it is time to trigger the corresponding ending.

Figure 23: In scene 11, I create a variable to access the despair points that were logged in dialogue.html. When the opened door is clicked, I use a conditional to open the corresponding ending's HTML file based on the amount of accumulated despair points.

Figure 24: The ending gameplay slightly differs in that there is a segment where clicking on particular objects progressed the gameplay; however, due to the way that the images were imported, clicking anywhere would advance the plot. Regardless, new classes were applied to all the text to make it visible against the lighter backgrounds, and the text triggered by the objects were isolated in their own classes.

Prototyping/Building Process (pg 6 of 7):
View original code [here](#).

```

120 let collected = 0
121 let hasCouch = false
122 let hasPlant = false
123 let hasPhoto = false
124
125 //Couch
126 couch.addEventListener("click", async () => {
127   ...if (!hasCouch) {
128     ...collected += 1;
129     ...tx_s122.style.display = "block";
130     ...couch.style.pointerEvents = "none";
131   }
132 });
133
134 tx_s122.addEventListener("click", async () => {
135   ...tx_s122.style.display = "none";
136   ...if (collected == 3) {
137     ...tx_s125.style.display = "block"
138   };
139 });
140
141 //Plant
142 plant.addEventListener("click", async () => {
143   ...if (!hasPlant) {
144     ...collected += 1;
145     ...tx_s123.style.display = "block";
146     ...plant.style.pointerEvents = "none";
147   }
148 });
149
150 tx_s123.addEventListener("click", async () => {
151   ...tx_s123.style.display = "none";
152   ...if (collected == 3) {
153     ...tx_s125.style.display = "block"
154   };
155 });
156
157 //Photo
158 photo.addEventListener("click", async () => {
159   ...if (!hasPhoto) {
160     ...collected += 1;
161     ...tx_s124.style.display = "block";
162     ...photo.style.pointerEvents = "none";
163   }
164 });
165
166 tx_s124.addEventListener("click", async () => {
167   ...tx_s124.style.display = "none";
168   ...if (collected == 3) {
169     ...tx_s125.style.display = "block"
170   };
171 });

```

Figures 25 & 26: To trigger each dialogue option individually, I created new variables to store whether or not a dialogue was triggered. Once a dialogue was triggered, the variable would change to true, and it would no longer be able to activate. A counter would keep track of how many dialogue options have been triggered, and when all three have been triggered, the ending sequence would play. The game would then navigate back to mirror.html for a full loop.

Figures 25 & 26: Setting flags for each “phase” of the ending

Prototyping/Building Process (pg 7 of 7):
View original code [here](#).

Conclusion / Reflection

My hopes for this piece were to make the player reflect on themselves and reassess their reliance on external validation and explanation on their own selves. When my peers tested the game for the first time, I was surprised by how people truly projected their own experiences onto the narrator's analyses. From what I have seen, I do think that I was successful in achieving my goal.

This game is meant to be almost ironic in that it critiques itself as a genre. In some of the dialogue options, the narrator hints at or even explicitly states that it should not be trusted over the player themselves. Despite that, players were still able to see truth in the narrator's assumptions of them. The narrator character is meant to be an entity that is meant to be uncomfortable and simultaneously trusted and disliked. They are seemingly peering into the core of the player, but at times its over confident tone is ridiculous and the player remembers that they are placing too much importance on what a program has to say about their identity.

My main takeaway that I hope players understood was that they will always understand themselves better than what anyone else and that self discovery will inevitably be uncomfortable.

This project was my first game, and I wrote, drew, and coded every piece of it myself, so I am happy with how it turned out. I hope that everyone who was able to experience this piece, either by playing it themselves or watching a video, had a good time and had something interesting to think about.

Thank you!

Links

Code	https://github.com/v3emily/the-self-in-the-mirror-the-mirror-in-the-self
Final video	https://drive.google.com/file/d/19jQse3s74qZ-5l8ki9sLxHcURq5izffG/view?usp=drive_link

Sources

If you used tutorials, Stack Overflow code, or other third-party materials, you should cite them here! It's good practice and this way you'll be able to find it again in the future :-)

If there are sections of your code that are from third-party sources, try to directly cite them in your code using a *//comment*, too.

JavaScript Stopwatch Tutorial	https://tinloof.com/blog/how-to-build-a-stopwatch-with-html-css-js-react-part-2
Enter key detection	https://coreui.io/answers/how-to-detect-enter-key-in-javascript/#:~:text=Check%20for%20event,.to%20detect%20Enter%20key%20presses.&text=This%20code%20listens%20for%20all,form%20or%20triggering%20a%20search.