

Table of Content:

- P1: Cover Page
- P2: Abstract
- P3: Research & Concept
- P4: Experiments with PEmbroider
- P5: Design (Procreate)
- P6: Data Scraping (Jupyter Notebook)
- P7: Embroidery (Processing)
- P8: Final Result
- P9: Evaluation
- P10: References

Data-Animal Remnant

An embroidery piece driven by Big data

Canace CHEN

(Turbosquid, 2015)

Abstract

"Data-Animal Remnant" is an embroidery project driven by big data, investigating the convergence of digital life, de-extinction, and the transformative power of large-scale data analysis. Employing the `processing.embroider` library and `PEmbroiderGraphics`, the artwork showcases the potential of big data in creative expression.

The project incorporates two primary data sources: a comprehensive website featuring endangered animals worldwide and an API from Steam, a popular gaming platform. By extracting animal groups and quantifying endangered species from the first source, the project establishes connections between these groups and the gaming world represented on Steam.

Through the interplay of embroidery and big data, "Data-Animal Remnant" prompts contemplation on the evolving boundaries between animals and machines in the digital age. It challenges traditional notions of animals as solely physical beings, as digitalization extends beyond a mere recording of their physical existence (Adams, 2020).

Moreover, the project raises thought-provoking questions about the authenticity of digitized creatures. While advanced technologies enable the resurrection of extinct animals like dinosaurs in virtual worlds, "Data-Animal Remnant" questions whether these recreations truly embody the essence of the original species.

"Data-Animal Remnant" underscores the significance of big data in reshaping our understanding of animals and their relationship with the digital realm. By bridging embroidery and data-driven insights, the project invites viewers to reflect on the ethical considerations, blurred boundaries, and profound impacts of big data on our perception of the animal kingdom and the rapidly evolving landscape of technology.

Research & Concept

Research:

Technological progress has empowered ecologists to study the impact of human development on ecosystems using electronic tools. These tools enable the generation, storage, sharing, and analysis of large quantities of data, revolutionizing ecological research.

Previously, wildlife tracking relied on radio-tracking and offered limited data points. However, the introduction of GPS recorders, accelerometers and other bio-logging tools in the early 2000s ushered in the era of "big data" in wildlife tracking. This enabled researchers to gather extensive information on the movements of marine megafauna in the ocean (Grémillet et al., 2022).

Furthermore, a groundbreaking technique called environmental DNA (eDNA) metabarcoding has emerged. This approach utilizes DNA fragments present in the environment to identify individuals, species, and communities. By studying eDNA, researchers can track organisms, explore their distributions, and examine changes in biodiversity across different time periods and locations (Taberlet et al., 2018).

These technological advancements have significantly advanced ecological understanding, providing scientists with unparalleled access to vast amounts of data. They have opened up new avenues for research, facilitating comprehensive exploration and conservation of the natural world.

Concept:

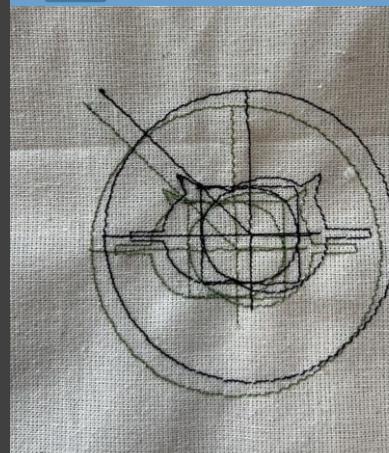
"Data-Animal Remnant" derives its name from the idea of using data remnants from different sources, including endangered animals and video games, to create an embroidered reflection on the coexistence of digital life and the potential for de-extinction.

From the first data source, animal groups and their respective numbers of endangered animals were extracted. These numbers determined the density of the animals represented in the embroidery. Higher numbers resulted in lower density, highlighting the alarming disappearance of these creatures. The second data source, the number of games related to each animal group on Steam, determined the frequency of each animal's appearance in the embroidery. The more games associated with a specific animal group, the greater the representation of that animal.

The project delves into the concept of digital life and de-extinction, contemplating the authenticity and identity of digitally resurrected creatures. As technology continues to advance, animals are being digitized for purposes ranging from military applications to animal conservation. The project prompts us to question whether the virtual representation of an extinct animal truly captures its essence or merely manifests as a human-made artifact.

Experiments with PEmbroider

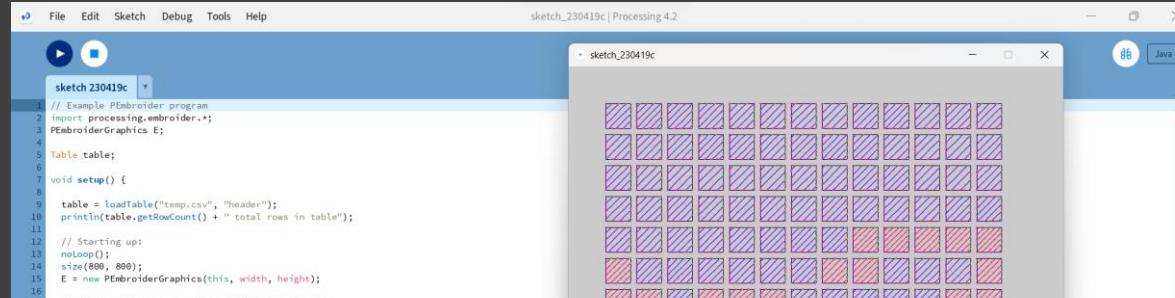
Experiment 1:



Embroidery Video:
<https://drive.google.com/file/d/1TjnakiAS9Zbn1Arr5RD9XChwx5LajI9/view?usp=sharing>

the pattern), `E.update(true, true, false)` would display it on screen. When the pattern is ready to be embroidered, use `E.endDraw()` to save the embroidery file. `sketchPath(".pes")` and `E.setPath()` are used to set the file path.

Experiment 2:



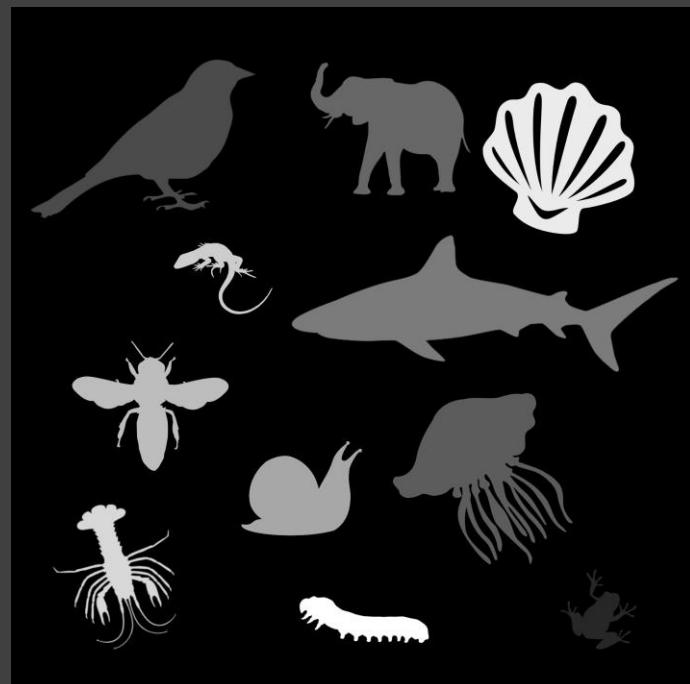
The image shows the Processing IDE interface. On the left, the code for a sketch titled 'sketch 230419c' is displayed. The code is a Processing sketch that reads a CSV file 'temp.csv', initializes a PEmbroiderGraphics object, and then uses a for loop to fill a 10x10 grid with colored squares (purple, pink, red) based on the data from the CSV file. The sketch window on the right shows the resulting 10x10 grid of colored squares.

```
// Example PEmbroider program
1 import processing.embroider.*;
2 PEmbroiderGraphics E;
3
4 Table table;
5
6 void setup() {
7
8   table = loadTable("temp.csv", "header");
9   println(table.getRowCount() + " total rows in table");
10
11  // Starting up:
12  noLoop();
13  size(600, 600);
14  E = new PEmbroiderGraphics(this, width, height);
15
16  // change this line you want a different file type
17  String outputFilePath = sketchPath("filename.pes");
18  E.setPath(outputFilePath);
19  E.beginDraw();
20  E.hatchMode(E.PARALLEL);
21  E.clear();
22
23  int count = 0;
24  int rowLength = 13;
25  E.hatchMode(E.PARALLEL);
26  E.fill(0,0,0);
27
28  E.beginComposite();
29  int spacing = 60;
30  for (Table row : table.rows()) {
31    int i = count % rowLength;
32    int j = floor(count/rowLength);
33
34    E.fill(0,0,0);
35    E.hatchMode(E.PARALLEL);
36    E.fill(0,0,0);
37
38    E.beginComposite();
39    E.fill(0,0,0);
40    E.hatchMode(E.PARALLEL);
41    E.fill(0,0,0);
42
43    E.beginComposite();
44    E.fill(0,0,0);
45    E.hatchMode(E.PARALLEL);
46    E.fill(0,0,0);
47
48    E.beginComposite();
49    E.fill(0,0,0);
50    E.hatchMode(E.PARALLEL);
51    E.fill(0,0,0);
52
53    E.beginComposite();
54    E.fill(0,0,0);
55    E.hatchMode(E.PARALLEL);
56    E.fill(0,0,0);
57
58    E.beginComposite();
59    E.fill(0,0,0);
60    E.hatchMode(E.PARALLEL);
61    E.fill(0,0,0);
62
63    E.beginComposite();
64    E.fill(0,0,0);
65    E.hatchMode(E.PARALLEL);
66    E.fill(0,0,0);
67
68    E.beginComposite();
69    E.fill(0,0,0);
70    E.hatchMode(E.PARALLEL);
71    E.fill(0,0,0);
72
73    E.beginComposite();
74    E.fill(0,0,0);
75    E.hatchMode(E.PARALLEL);
76    E.fill(0,0,0);
77
78    E.beginComposite();
79    E.fill(0,0,0);
80    E.hatchMode(E.PARALLEL);
81    E.fill(0,0,0);
82
83    E.beginComposite();
84    E.fill(0,0,0);
85    E.hatchMode(E.PARALLEL);
86    E.fill(0,0,0);
87
88    E.beginComposite();
89    E.fill(0,0,0);
90    E.hatchMode(E.PARALLEL);
91    E.fill(0,0,0);
92
93    E.beginComposite();
94    E.fill(0,0,0);
95    E.hatchMode(E.PARALLEL);
96    E.fill(0,0,0);
97
98    E.beginComposite();
99    E.fill(0,0,0);
100   E.hatchMode(E.PARALLEL);
101   E.fill(0,0,0);
102
103   E.endComposite();
104   E.endComposite();
105   E.endComposite();
106   E.endComposite();
107
108   E.endComposite();
109   E.endComposite();
110
111   E.endComposite();
112   E.endComposite();
113
114   E.endComposite();
115   E.endComposite();
116
117   E.endComposite();
118   E.endComposite();
119
120   E.endComposite();
121   E.endComposite();
122
123   E.endComposite();
124   E.endComposite();
125
126   E.endComposite();
127   E.endComposite();
128
129   E.endComposite();
130   E.endComposite();
131
132   E.endComposite();
133   E.endComposite();
134
135   E.endComposite();
136   E.endComposite();
137
138   E.endComposite();
139   E.endComposite();
140
141   E.endComposite();
142   E.endComposite();
143
144   E.endComposite();
145   E.endComposite();
146
147   E.endComposite();
148   E.endComposite();
149
150   E.endComposite();
151   E.endComposite();
152
153   E.endComposite();
154   E.endComposite();
155
156   E.endComposite();
157   E.endComposite();
158
159   E.endComposite();
160   E.endComposite();
161
162   E.endComposite();
163   E.endComposite();
164
165   E.endComposite();
166   E.endComposite();
167
168   E.endComposite();
169   E.endComposite();
170
171   E.endComposite();
172   E.endComposite();
173
174   E.endComposite();
175   E.endComposite();
176
177   E.endComposite();
178   E.endComposite();
179
180   E.endComposite();
181   E.endComposite();
182
183   E.endComposite();
184   E.endComposite();
185
186   E.endComposite();
187   E.endComposite();
188
189   E.endComposite();
190   E.endComposite();
191
192   E.endComposite();
193   E.endComposite();
194
195   E.endComposite();
196   E.endComposite();
197
198   E.endComposite();
199   E.endComposite();
200
201   E.endComposite();
202   E.endComposite();
203
204   E.endComposite();
205   E.endComposite();
206
207   E.endComposite();
208   E.endComposite();
209
210   E.endComposite();
211   E.endComposite();
212
213   E.endComposite();
214   E.endComposite();
215
216   E.endComposite();
217   E.endComposite();
218
219   E.endComposite();
220   E.endComposite();
221
222   E.endComposite();
223   E.endComposite();
224
225   E.endComposite();
226   E.endComposite();
227
228   E.endComposite();
229   E.endComposite();
230
231   E.endComposite();
232   E.endComposite();
233
234   E.endComposite();
235   E.endComposite();
236
237   E.endComposite();
238   E.endComposite();
239
240   E.endComposite();
241   E.endComposite();
242
243   E.endComposite();
244   E.endComposite();
245
246   E.endComposite();
247   E.endComposite();
248
249   E.endComposite();
250   E.endComposite();
251
252   E.endComposite();
253   E.endComposite();
254
255   E.endComposite();
256   E.endComposite();
257
258   E.endComposite();
259   E.endComposite();
260
261   E.endComposite();
262   E.endComposite();
263
264   E.endComposite();
265   E.endComposite();
266
267   E.endComposite();
268   E.endComposite();
269
270   E.endComposite();
271   E.endComposite();
272
273   E.endComposite();
274   E.endComposite();
275
276   E.endComposite();
277   E.endComposite();
278
279   E.endComposite();
280   E.endComposite();
281
282   E.endComposite();
283   E.endComposite();
284
285   E.endComposite();
286   E.endComposite();
287
288   E.endComposite();
289   E.endComposite();
290
291   E.endComposite();
292   E.endComposite();
293
294   E.endComposite();
295   E.endComposite();
296
297   E.endComposite();
298   E.endComposite();
299
299 year 2018 temp 0.85
300 year 2019 temp 0.98
301 year 2020 temp 1.02
302 year 2021 temp 0.85
303 year 2022 temp 0.89
```

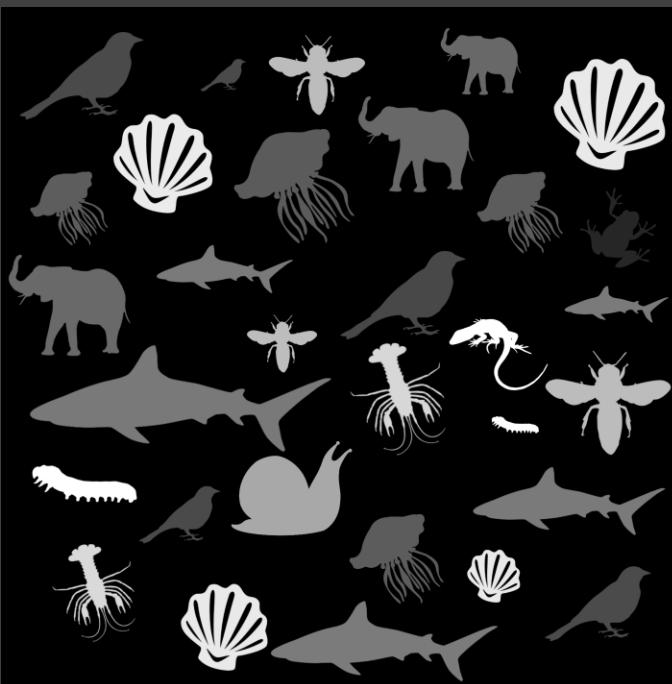
The second experiment was using the PEmbroider Library for data visualisation. The data used was from a .csv file containing smoothed and unsmoothed data on temperature from 1880 to 2022. The year and unsmoothed temperature columns were used. Each square represents a year, and its colour and density were determined by the temperature in that year. When $\text{temp} < 0.0$, the colour would be blue and if $\text{temp} > 0.0 \ \&\& \ \text{temp} < 0.5$, the colour would be red. In addition, the temp data was mapped from $(-0.47, 1.02)$ to $(10.0, 2.0)$ to affect the density of the stitching. This means as the temperature gets higher, the square would be more densely stitched. From the result pattern, it is obvious that the temperature has greatly increased from 1880 to 2020.

Design (Procreate)

Design 1:



Design 2:



The 2nd design kept the use of the number of endangered animals in each animal group to control the density. However, it added multiple numbers of each animal. The number of each animal was determined by the number of games contains/about the animal groups on Steam, as these games has created 'digital copies' of them in the virtual world. Each animal will be put in a random position within the frame without overlapping. These changes gives the pattern a fuller and more aesthetic look.

As there will be two data sources used in the pattern, there will be two variables impacting the pattern's look. The first design had the number of endangered animals in each animal group to control the density; larger number of endangered animals' results in lower density, to show these animals are disappearing. The number of games contains/about the animal groups on Steam to control the size of each animal, as they appear in the digital world more commonly.

Animals design:
I drew all these animals for each animal group.



Data Scraping (Jupyter Notebook)

Data Source 1: Worldwide Endangered Animals (website)

The first data source is a website containing worldwide endangered animals. The data was scraped by using `requests.get(web_url)`.

The heading and body rows are found by using BeautifulSoup `find_all(tag)`. The website contains 26 tables (a-z), therefore a nested for loop was used to loop through all tables.

After getting only the Group column of all tables, a new table was created with the all the groups (Group) and the number of endangered animals it has (Count). The new table was exported as a .csv file using pandas DataFrame.

The second data source is an official API from Steam and managed by Team Fortress. The API, GetAppList, contains a full list of every publicly facing program in the Steam store/library. The data was scraped by using `requests.get(api_url)`.

The all the data was load as a Json file. To get game that relates to each group of animal, a list of 10 most related words to the animal group was created to check if the games' names contain any of these words. If so, append them to the `filtered_apps` list.

After finding all games relating to each animal groups, a new table was created with the all the groups (Group) and the number of games relate to it (Games). The new table was exported as a .csv file using pandas DataFrame.

Embroidery (Processing)

Load and use data:

```
37 table = loadTable("AnimalSteam.csv", "header");
38 imageCounts = new int[numImages];
39 //println(table.getRowCount() + " total rows in table")
40
41 //get data from the animal games on steam table
42 for (int i = 0; i < numImages; i++) {
43   TableRow row = table.getRow(i);
44   // use the number of games for each group as number of
45   int value = row.getInt("Games");
46   // map games value to 1 (appear once) to 6 (appear 6
47   // as the range is too big and there are big gaps
48   // (eg: between insects(7030) and mammals(2496, the closest
49   imageCounts[i] = int(map(value, 13, 7030/3, 1, 6));
50 }
51
52 // Generate a list to store the indices of the images
53 ArrayList<Integer> imageIndices = new ArrayList<Integer>();
54
55 // Add image indices based on the desired appearance counts
56 for (int i = 0; i < imageCounts[0]; i++) {
57   imageIndices.add(0); // Animal 1
58 }
59 for (int i = 0; i < imageCounts[1]; i++) {
60   imageIndices.add(1); // Animal 2
61 }
62 for (int i = 0; i < imageCounts[2]; i++) {
63   imageIndices.add(2); // Animal 3
64 }
```

Load table 1 and map the values in the Games column from (13, 7030/3) to (1, 6). Create an ArrayList to store record the number of images. Use for loop to set the total number of each animal images to the value mapped from data.

```
99 table2 = loadTable("EndangeredAnimalGroups.csv", "header");
100 densities = new float[numImages];
101 //println(table.getRowCount() + " total rows in table")
102
103 //get data from the endangered animal groups table
104 for (int i = 0; i < numImages; i++) {
105   TableRow row = table2.getRow(i);
106   // use the count for each group as density (the more
107   float value = row.getFloat("Count");
108   // map count value to 7.0 (least dense) to 2.0 (most
109   densities[i] = map(value, 2294.0, 1.0, 6.5, 2.0);
110 }
```

Load table 2 and map the values in the Counts column from (2294.0, 1.0) to (6.5, 2.0) and use mapped data as density.

Within screen and no overlap:

```
5 //create an ArrayList to store the postions of the animals
6 ArrayList<PVector> animals = new ArrayList<PVector>();
17 // Maximum number of attempts to find a non-overlapping
18 int maxAttempts = 50000;
```

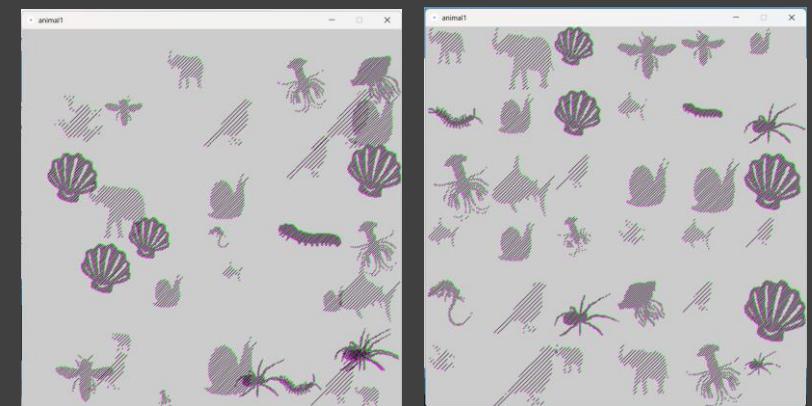
Create an ArrayList to record all position vector data. Set max attempt.

```
86 //try to find a position that does not overlap with other
87 while (overlapping && attempts < maxAttempts) {
88   // Generate random position within the canvas
89   x = int(random(width));
90   y = int(random(height));
91
92   // Check if the new circle overlaps with any existing
93   overlapping = checkOverlap(x, y, size);
94
95   //if not overlapping
96   if (!overlapping) {
97     //keep animals in the frame
98     x = constrain(x, size/2, width-size);
99     y = constrain(y, size/2, height-size);
100    //println(overlapping);
101    //println(x + " " + y);
102    // Add the circle position to the list
103    animals.add(new PVector(x, y));
104    println(animals);
105    // Draw the image
106    E.image(imgs[i], x, y, size, size);
107
108    break;
109  }
110  attempts++;
111 }
```

Use a while loop to try to find a non-overlapping position. If found, ensure the image stays in screen, add position to ArrayList and draw the image.

```
121 //check if the animals are overlapped
122 boolean checkOverlap(int x, int y, int size) {
123   // Check if the new circle overlaps with any existing
124   for (PVector animal : animals) {
125     float distance = dist(x, y, animal.x, animal.y);
126     if (distance < size) { // Minimum distance between
127       return true; // Overlapping
128     }
129   }
130   return false; // Non-overlapping
131 }
```

Check if images overlap.



The 1st pattern was from a discarded sketch. it draws the animals in random positions, but even with the checkOverlap function, the animals still overlaps. As the animals' sizes are random and the non-overlapping distance depends on the size of the closest animals, it is impossible to make sure they all maintain a perfect distance. Therefore, as show in the 2nd pattern, the animals are place in a 6*6 grid instead. This way it is easier to ensure the animals do not overlap.

```
136 // Display the images in the grid
137 int imageIndex = 0; // Index of the current image in the
138 for (int row = 0; row < gridSize; row++) {
139   for (int col = 0; col < gridSize; col++) {
140     int x = col * (width / gridSize) + 5; // X-coordinate of
141     int y = row * (height / gridSize); // Y-coordinate of
142     int size = int(random(60, width / gridSize + 15)); //set
143
144     // Get the current image index from the shuffled list
145     int currentImageIndex = imageIndices.get(imageIndex);
146
147     // Get the density for the current animal
148     E.hatchSpacing(densities[currentImageIndex]);
149
150     // Display the current animal
151     E.image(imgs[currentImageIndex], x, y, size, size);
152
153     // Move to the next image index
154     imageIndex++;
155   }
156 }
```

Using nested for loop to place animals on a grid.

Final Result



Video:

https://drive.google.com/file/d/1FYMA1CjdYaKFRKTr_BcS4UsIi9JF8J/view?usp=sharing

Full code:

<https://git.arts.ac.uk/21020295/Data-Animal-Remnant>



Evaluation

Self-evaluation:

In this project, I merged embroidery, big data, and digital life. I transformed data into tangible art with processing and embroidery. By extracting information from a website on endangered animals and the Steam API for game data, I explored the relationship between animal conservation, digital existence, and human influence.

Through embroidery, I visualized the declining presence of endangered animals. Their density in the artwork reflected their dwindling numbers. Additionally, the number of games related to each animal group determined their representation in the embroidery, showcasing the concept of digital life and de-extinction.

"Data-Animal Remnant" integrated art, data, and philosophical inquiry, urging reflection on digital life's impact on our perception of reality. It emphasized the fragility of endangered species and invited contemplation on our role in conservation efforts and the blurred boundaries between the natural and the human-made.

Possible improvement:

1. The method used in finding all games on Steam related to each animal groups needs improvement in its accuracy. As words appeared in the name of games do not necessarily relate to its content.
2. While the concept of using embroidery to represent the density of endangered animals and their digital counterparts is intriguing, there is room for further refinement in the execution of the artwork itself. Consider exploring different mediums and technologies that can effectively convey the intended message and evoke a stronger emotional response from viewers.
1. The project touches upon thought-provoking questions about the authenticity of digital representations and the nature of digital life. To strengthen this aspect, consider expanding on the philosophical inquiry through additional research, references, or engagement with relevant theories.

References

1. Adams, W. (2020). *Digital Animals*. <https://doi.org/10.17863/CAM.48779> (Accessed: 5 May 2023).
2. Alonso, N. M. (2021). PEmbroider: An Open Source Embroidery Library for Processing. Available at: <https://www.youtube.com/watch?v=3yKEUhSFStc> (Accessed: 19 May 2023).
3. Cameron, A. (2023). *Workshop: Generative Embroidery with PEmbroider*. Available at: [Creating Digital Embroidery with PEmbroider | CCI Wiki \(arts.ac.uk\)](https://arts.ac.uk/creating-digital-embroidery-with-pembroider/) (Accessed: 12 June 2023).
4. Cameron, A. (2022). *Creating Digital Embroidery Designs with PEmbroider*. Available at: [Workshop: Generative Embroidery with PEmbroider | CCI Wiki \(arts.ac.uk\)](https://arts.ac.uk/creating-digital-embroidery-designs-with-pembroider/) (Accessed: 12 June 2023).
5. Earth's Endangered Creatures (2023). Worldwide Endangered Animal List. Available at: <http://www.earthsendangered.com/list.asp> (Accessed: 1 Jun 2023).
6. Grémillet, D. et al. (2022). *Big data approaches to the spatial ecology and conservation of marine megafauna*. Available at: [Big data approaches to the spatial ecology and conservation of marine megafauna | ICES Journal of Marine Science | Oxford Academic \(oup.com\)](https://doi.org/10.1080/02677038.2022.1480001) (Accessed: 20 May 2023).
7. Koech, K. E. (2020). *Web Scraping 1: Scraping Table Data*. Available at: <https://towardsdatascience.com/web-scraping-scraping-table-data-1665b6b2271c> (Accessed: 23 May 2023).
8. Python Pool (no date). What Causes “iopub data rate exceeded” Problem and How to Fix it. Available at: [What Causes “iopub data rate exceeded” Problem and How to Fix it \(pythonpool.com\)](https://pythonpool.com/article/iopub-data-rate-exceeded/) (Accessed: 4 Jun 2023)
9. Taberlet, P. et al. (2018). *Environmental DNA: For Biodiversity Research and Monitoring*. Kettering: Oxford University Press.
10. Team Fortress (2021). *WebAPI/GetAppList*. Available at: [WebAPI/GetAppList - Official TF2 Wiki | Official Team Fortress Wiki](https://tf2.fandom.com/wiki/WebAPI/GetAppList) (Accessed: 28 May 2023).
11. Turbosquid (2015). *Jurassic Park Raptor 3d Model*. Available at: [jurassic park raptor 3d model \(turbosquid.com\)](https://www.turbosquid.com/3d-models/jurassic-park-raptor-3d-model/1113000) (Accessed: 7 Jun 2023).