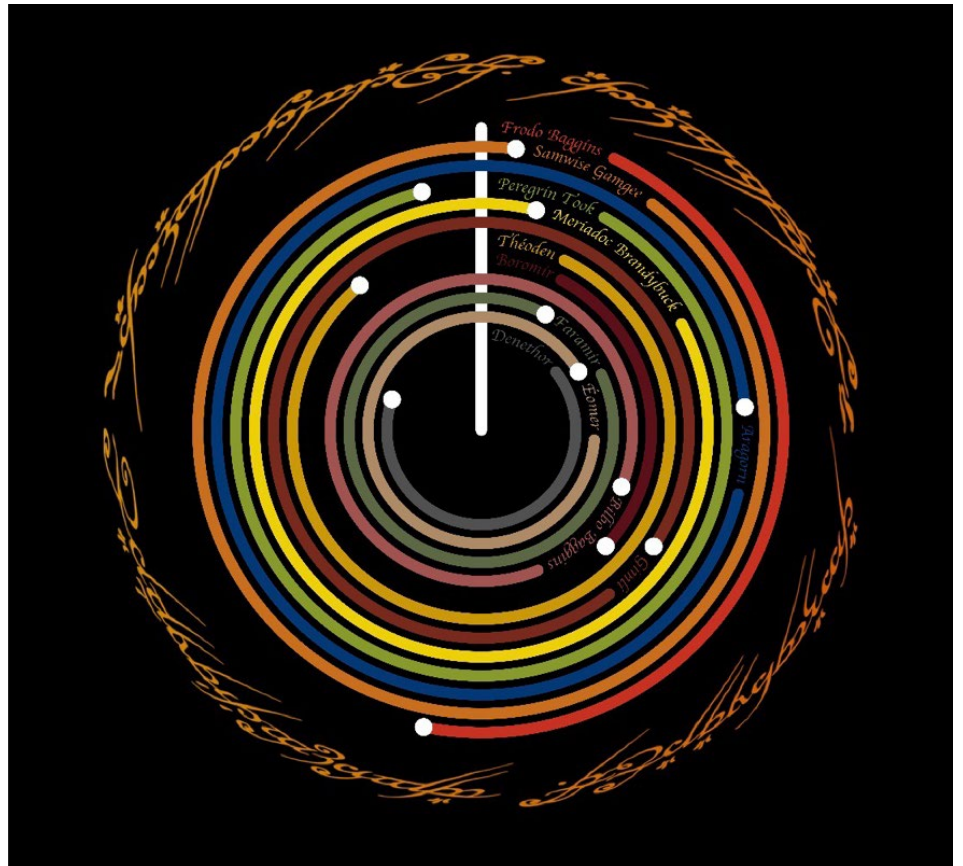


Lifespans In Middle Earth Data Art - Report

Final Results

The final Data Art piece, named Lifespans In Middle Earth, successfully portrays the lifespans of various characters. It does this by displaying each character as a circular arc, each with a unique colour for satisfying distinguishment, the colours were mostly chosen based on the association with the given character. Each arc has a white ellipse that stays with the end of the arc, indicating the end of a character's life.

A white line is generated from the centre of the piece that protrudes upwards to the edge of the coloured arcs, referencing the average age of death for a character's race. When a character's circular arc extends past the white line, that means they lived past the average lifespan of their race, and it can be seen by how much on the piece.



Also included, is the names of each character labelling their respective circles, the letters are coded to curve along with the circular arcs. Upon running the code, the art piece will iteratively be drawn over a period of time, giving the piece a dynamic dimension for potential online viewing.

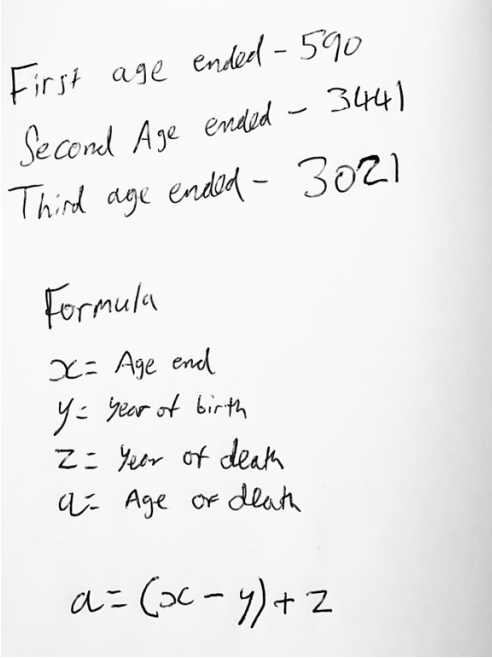
To enhance the Tolkien theme the outside of the circular arcs shows an image of the inscription from the One Ring, written in the fiery letters of Black Speech, this image was sketched in Photoshop and imported into Processing. This piece was also inspired by the iconography of the Eye of Sauron; hence the art piece appears eye shaped and similarly 'wreathed in flame' in a way.

Techniques Used

To create the Data Art piece, many methods were employed:

Data processing:

Before the data could be used, it had to be properly adapted for the processing scripts. This was done by calculating the age of death for each of the selected characters from the dataset, this had to be done since the original dataset did not have values for age of death. This was completed by subtracting the character's year of birth from their year of death, although this couldn't be done for all characters due to the changeover of the Third Age to the Fourth Age of Middle Earth, when the year would change from 3021 TA (Third Age) to 1 FO (Fourth Age), this was rectified through use of this simple formula:



First age ended - 590
Second Age ended - 3441
Third age ended - 3021

Formula

x = Age end
 y = year of birth
 z = year of death
 a = Age of death

$$a = (x - y) + z$$

Once the age of death had been determined for each character, some more data that was not included in the original dataset had to be retrieved, that being the average age of death for each race, this data was obtained through internet sources such as [LOTR Fanon - Fandom](#) and [Tolkien Gateway](#). Once all the data was collected it was organised into a separate spreadsheet:

Character	Race	Age of Death	Average Life Span	Angle		Colour (RGB)
Frodo Baggins	Hobbit	53	100	100.8	Red/Brown	200, 48, 35
Samwise Gamgee	Hobbit	102	100	277.2	Orange/Ginger	200, 111, 29
Aragorn	Men/Dúnedain	210	170	354.7	Black/Blue	0, 57, 117
Peregrin Took	Hobbit	96	100	255.6	Green/Brown	135, 154, 44
Meriadoc Brandybuck	Hobbit	104	100	284.4	Yellow/Orange	238, 206, 0
Gimli	Dwarf	262	195	393.7	Burgundy/Brown	120, 42, 29
Théoden	Men	71	80	229.5	Blonde/Red	203, 152, 0
Boromir	Men/Dúnedain	41	111	43	Red/Black	91, 16, 26
Bilbo Baggins	Hobbit	131	100	381.6	Red/White	161, 83, 81
Faramir	Men/Dúnedain	120	111	299.2	Green/Grey	92, 105, 68
Éomer	Men	93	80	328.5	Blonde/Brown	176, 139, 103
Denethor	Men/Dúnedain	89	111	198.6	Grey/Black	81, 81, 81

The angle quadrants in Processing had an offset of 90° so the positive y-axis started at -90° instead of 0°, to account for this when calculating the necessary angles for each character, the formula of $\theta^\circ = (360 * (x / y)) - 90$ was used where x is the age of death, and y is the average lifespan, the subsequent values would be ready to use within Processing.

The characters were also sorted highest to lowest based on their screen-time in the films.

A colour palette was also selected in the spreadsheet, signifying what the colours of the character circles would be, labelled with their RGB number for simple transfer to Processing.

Arcs:

The primary shapes used in the art piece were arcs, in Processing, the arc() function was used to accomplish this. The angles made in the spreadsheet were assigned to the *character_nameEnd* variables since this is the angle where the arc will stop. The colours for the arcs were carried over from the spreadsheet's RGB values and assigned to the *character_nameR*, *character_nameG*, *character_nameB* variables respectively. To arrange the arcs so that the more prominent characters appear on the outside, the individual arcs were scaled linearly.

```

20 //Frodo data input
21 float frodoStart = -64;
22 float frodoEnd = 101;
23 float frodoCurrent = frodoStart;
24 int frodoR = 200;
25 int frodoG = 48;
26 int frodoB = 35;
27 String frodo = "Frodo Baggins";
28 float frodoRad = 3160;
29
30 //Sam data input
31 float samStart = -53;
32 float samEnd = 277;
33 float samCurrent = samStart;
34 int samR = 200;
35 int samG = 111;
36 int samB = 29;
37 String sam = "Samwise Gamgee";
38 float samRad = 2960;

// Start angle for Frodo
// End angle for Frodo
// Initialise Frodo angle current value and sets to Frodo start
// Frodo arc RGB red value
// Frodo arc RGB green value
// Frodo arc RGB blue value
// String to store full character name for text display
// Radius of Frodo arc

// Repeat for all characters
//
//
//
//
//

```

```

154 void draw() { // Begin draw
155
156     background(0); // Set background to black
157
158     //Ring image
159     image(blackspeech, 1000, 1000, width * 0.8, height * 0.8); // Paste black speech image and set size and offset
160
161     //Frodo
162     if (frodoCurrent < frodoEnd) { // This will execute every draw cycle while the current Frodo arc angle is less than the end arc angle value
163         // so that it will iteratively draw the arc until it reaches the end point
164         stroke(frodoR, frodoG, frodoB); // Sets the stroke colour to the assigned colour for Frodo
165         noFill(); // Disables fill
166         arc(5000, 5000, 6400, 6400, radians(frodoStart), radians(frodoCurrent), OPEN); // Begin drawing arc at frodoStart and finish at frodoCurrent
167         frodoCurrent++; // Increments the frodoCurrent value for the next iteration
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218 //Pippin
219 if (pipCurrent < pipEnd) {
220
221     stroke(pipR, pipG, pipB);
222     noFill();
223     arc(5000, 5000, 5200, 5200, radians(pipStart), radians(pipCurrent), OPEN); // Subsequent characters have a smaller arcs
224     pipCurrent++;
225
226     stroke(255, 255, 255);
227     fill(255, 255, 255);
228     ellipse(5000 + cos(radians(pipCurrent)) * (pipRad + 40), 5000 + (sin(radians(pipCurrent)) * (pipRad + 40)), 70, 70);
229
230 }
231
232

```

White Line:

The central white line was coded similarly to the arcs, using lineStart and lineEnd variables to define where it should begin and finish.

```

13 float lineStart = 5000; // Start y point for middle line
14 float lineEnd = 1800; // End y point for middle line
15 float lineCurrent = lineStart; // Initialise middle line current position and sets to line start
16 int lineR = 255; // Middle line RGB red value
17 int lineG = 255; // Middle line RGB green value
18 int lineB = 255; // Middle line RGB blue value
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Animated Generation:

This art piece is able to generate iteratively over a span of time through the use of the draw() functionality, as well as if() statements encompassing the draw commands. The if() statement condition would be – if(*character_nameCurrent* is less than *character_nameEnd*) then the shapes, whether an arc or line, would be drawn from the *character_nameStart* position to the *character_nameCurrent*, which would begin equal to the *character_nameStart* variable, but would then increment by one after each draw, therefore, each time the draw() function is run, the shape would be become more complete, until it reaches the *character_nameEnd* value.

```

154 void draw() { // Begin draw
155
156     background(0); // Set background to black
157
158     //Ring image
159     image(blackspeech, 1000, 1000, width * 0.8, height * 0.8); // Paste black speech image and set size and offset
160
161     //Frodo
162     if (frodoCurrent < frodoEnd) { // This will execute every draw cycle while the current Frodo arc angle is less than the end arc angle value
163         // so that it will iteratively draw the arc until it reaches the end point
164         stroke(frodoR, frodoG, frodoB); // Sets the stroke colour to the assigned colour for Frodo
165         noFill(); // Disables fill
166         arc(5000, 5000, 6400, 6400, radians(frodoStart), radians(frodoCurrent), OPEN); // Begin drawing arc at frodoStart and finish at frodoCurrent
167         frodoCurrent++; // Increments the frodoCurrent value for the next iteration
168
169         stroke(255, 255, 255); // Sets stroke colour to white
170         fill(255, 255, 255); // Enables fill and sets fill colour to white
171         ellipse(5000 + cos(radians(frodoCurrent)) * (frodoRad + 40), 5000 + (sin(radians(frodoCurrent)) * (frodoRad + 40)), 70, 70); // Creates an ellipse at the end of the current position of the arc
172
173     }
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439 //Line
440 if (lineCurrent > lineEnd) {
441
442     stroke(lineR, lineG, lineB);
443     line(5000, lineStart, 5000, lineCurrent);
444     lineCurrent = lineCurrent - 10;
445
446 }
447
448 if (lineCurrent == lineEnd) {
449
450     stroke(lineR, lineG, lineB);
451     line(5000, lineStart, 5000, lineEnd);
452
453 }
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Once this would happen the entire shape would disappear due to the condition turning false, to fix this, a different if() statement was introduced with the condition of if(*character_nameCurrent* is equal to *character_nameEnd*), it would then draw the complete shape, since this condition will always be true as long as the image is complete, the image will remain.

```

175 if (frodoCurrent == frodoEnd) { // This will execute once the frodoCurrent value has reached the frodoEnd value/when the arc has reached the desired point
176
177 stroke(frodoR, frodoG, frodoB); // Sets stroke colour to the assigned colour for Frodo
178 noFill();
179 arc(5000, 5000, 6400, radians(frodoStart), radians(frodoEnd), OPEN); // Draws the complete arc from frodoStart to frodoEnd/will remain unchanged through future iterations while frodoCurrent is equal to frodoEnd
180
181 stroke(255, 255, 255); // Sets stroke colour to white
182 fill(255, 255, 255); // Enables fill and sets fill colour to white
183 ellipse(5000 + cos(radians(frodoEnd)) * (frodoRad + 40), 5000 + (sin(radians(frodoEnd)) * (frodoRad + 40)), 70, 70); // Creates an ellipse at the end of the arc.
184

```

To enhance the visual aesthetic, as well as to aid the viewer in distinguishing the end points for the lines, a white ellipse was added to the end of each character line, the ellipse follows the same animation as the line and remains at the end of it through the use of a mathematical function that can calculate a point on a circle, given the circle radius and current angle: $x = r(\cos(\theta^\circ))$ and $y = r(\sin(\theta^\circ))$.

```

189 stroke(255, 255, 255); // Sets stroke colour to white
190 fill(255, 255, 255); // Enables fill and sets fill colour to white
191 ellipse(5000 + cos(radians(frodoCurrent)) * (frodoRad + 40), 5000 + (sin(radians(frodoCurrent)) * (frodoRad + 40)), 70, 70); // Creates an ellipse at the end of the current position of the arc.
192
193

```

To retain the image of the ellipse when the image is fully generated, the ellipse is drawn again in the new if() statement similarly to the arc:

```

181 stroke(255, 255, 255); // Sets stroke colour to white
182 fill(255, 255, 255); // Enables fill and sets fill colour to white
183 ellipse(5000 + cos(radians(frodoEnd)) * (frodoRad + 40), 5000 + (sin(radians(frodoEnd)) * (frodoRad + 40)), 70, 70); // Creates an ellipse at the end of the arc

```

Curved Text:

One of the most challenging aspects of the art piece was to be able to curve the names of the characters with the arcs. To accomplish this, I found some code [online](#) and adapted it to work for each of the arcs in the art piece. The method consists of a loop that takes each letter of a given string and applies a transform to the individual letters, giving the effect that a word is curving. The letters would be transformed in reference to the radius of the arc, and the output word would appear to have the same curvature as the arc. For each character, the text position on the circumference had to be tweaked manually.

```

7 int i; // Iterator number variable for loops
8 float arcLength; // Variable to control rotation on letters
9 char currentChar; // Variable storing current letter
10 float w; // Variable for storing letter width
11 float theta; // Variable to store angles

```

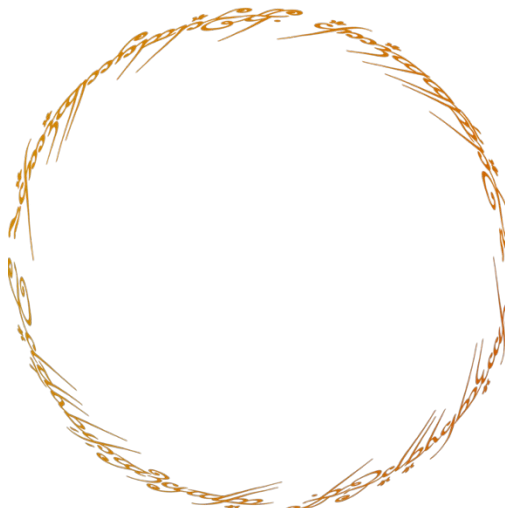
```

185 //FrodoText
186 noFill(); // Disables fill
187 stroke(frodoR, frodoG, frodoB); // Sets the stroke colour to the assigned colour for Frodo
188
189 arcLength = 0; // Resets arcLength to zero
190
191 for (i = 0; i < frodo.length(); i++) { // Loop to iterate through the length of the String text
192
193     currentChar = frodo.charAt(i); // Sets the current letter to the iterator i value
194     w = textWidth(currentChar); // Sets the letter width variable to the current letter width
195     arcLength += w/2; // Adds half the letter width to the arcLength
196
197     theta = PI * 1.02 + arcLength / frodoRad; // Sets the position of the text to the appropriate position on the circumference of the arc
198
199     pushMatrix(); // Pushes the current transformation matrix to the matrix stack
200
201     translate(frodoRad*cos(theta) + 5000, frodoRad*sin(theta) + 5000); // Places the text at the correct offset
202     rotate(theta + PI/2); // Rotates the text according to its position on the circumference
203
204     fill(frodoR, frodoG, frodoB); // Sets the text colour to the assigned colour for Frodo
205     translate(frodoRad, frodoRad); // Sets the text to the height and width of the radius to the correct position on the circumference
206     rotate(89.6); // Fine tuning rotation
207     text(currentChar, 0, 0); // Prints text
208     popMatrix(); // Pops the current transformation matrix off the matrix stack
209
210     arcLength += w/2; // Adds half the letter width to the arcLength once more
211 // Will repeat for all letters of the string
212
213 }

```

Black Speech Image:

The outer image of the Black Speech inscription that appears on the One Ring was sketched in photoshop by tracing over the official image and adding some gradient colour. The image was then curved in a ring shape so that it could surround the art piece, to give reference to the iconic One Ring, and also to the Eye of Sauron.



Critical Reflection

Overall, the Data Art piece has achieved the goal of presenting the lifespans of characters of Middle Earth. The data is clearly laid out with little clutter or overuse of colour, the colours themselves feel relevant to the theme. Once understood what is represented in the piece, the data can be clearly understood by the viewer.

Some aspects that could have been better would be the use of less monochrome colours per arc, for example, the colour could be enhanced using textures, to add more detail and intrigue to them. Another aspect to consider improving on would be the incorporation of more data, since the data presented in this piece is mostly one dimensional and does not encourage longer viewing, since the data can be read fairly quickly. A final element that could be better would be the use of more characters, since this would give more viewable data and hence make the piece more appealing.