Assignment 2: Evo Art and Procedural Textures

Harmandeep Mangat (hm15mx 6021109), Yujie Wang (yw14dc 5814637)

I. Introduction

Using GP to generate evolutionary arts and procedural textures has been a main attraction for some time now. While being able to create new and fancy art is definitely exciting, another use for GP is recreating an entire image in full detail. The purpose of this paper is to investigate the recreation of certain target images using a GP system. Multiple target images will be used to ensure that the GP system created will produce many recreated images to a somewhat high degree of accuracy. The following content will be discussing the results of an evo-art system with GP and how different languages effect the end result. Three experiments with their own three sub-experiments will be performed on three different target images, each varying in difficulty. The first experiment will use a basic set of languages and each proceeding sub-experiments will use more complicated languages designed for image processing. This system will take texture space coordinates X and Y as terminal nodes and evolve a formulae for R, G, and B, which then could be applied to recreate the target image. The system has three trees, each representing R, G, and B with the same function set, whose results are then evaluated and scored by the fitness function.

II. Experiment Details

A. Parameter Table

Crossover	90%
Mutation rate	10%
Max Depth	17
Tournament Size	7
Generation	51
Population size	3072
Elitism	10
Subtrees	3
Training set size	65536
Number of runs	10

TABLE I

B. Data Sources + Target Images + Setup + Misc

How to setup and run an experiment:

1) A target image must be loaded into the program under the read file function. Target images are recommended to be at least 256 by 256 pixels in area.
2) Under the Assignment2.params file, the parameters used by the GP system can be adjusted there. For this experiment, the parameters are mentioned above.
3) Inside the second ECJ directory, type 'make' into the command line to compile the program.
4) Followed by typing 'export CLASSPATH=$(pwd)/target/classes/' to target the executable.
5) Type 'java ec.Evolve -file target/classes/ec/app/Assignment2/Assignment2.params' to run the program.

Three images will be used for this experiment. They vary in difficulty. Figure 2 is the easiest due to its uniformity and singular style coloring, followed by figure 3 which starts to blend and distort its color borders. Finally, figure 1 is supposed to be the hardest because it deals with facial recognition. The images are shown as follows along with a citation of a data source used to get more advanced functions for procedural textures.

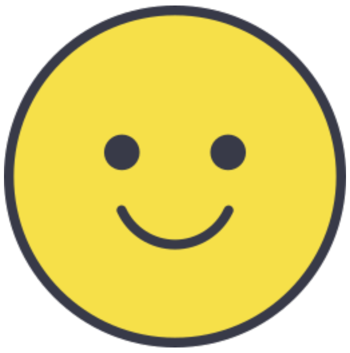
Fig. 1: Smiley Face Target Image

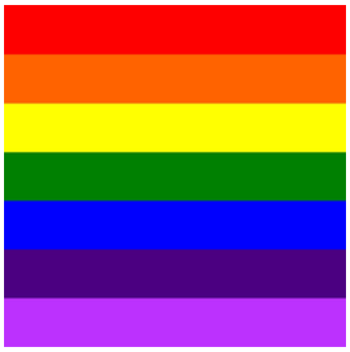
Fig. 2: Rainbow Target Image

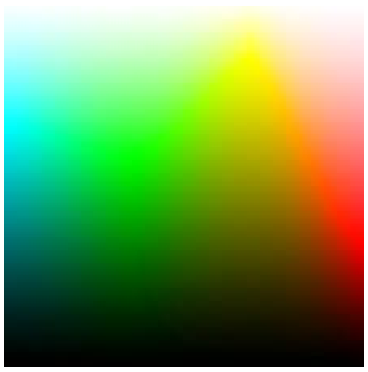
Fig. 3: Gradient Target Image

Smiley Face. (n.d.). icons8. Retrieved March 18, 2022, from https://icons8.com/illustrations/smiley-face.

Gando, F. (n.d.). Rainbow Theme. Tooliphone. Retrieved March 18, 2022, from https://iskin.tooliphone.net/ios-theme-without-jailbreak-theme-8558-user-12969.

Gradient of Colors. (n.d.). gstatic. Retrieved March 18, 2022, from https://encrypted-tbn0.gstatic.com/images

Miller, J. F. (2011). Introduction to evolutionary computation and Genetic Programming. Cartesian Genetic Programming, 1-16. https://doi.org/10.1007/978-3-642-17310-3_1

C. GP Language

Parameter - An user specified integer value
Input1 - Node 1
Input2 - Node 2

1) *Add:* $input1 + input2$
2) *Subtract:* $input1 - input2$
3) *Multiply:* $input1 * input2$
4) *Divide:* $input2/input2$ (if input2→0 then input2→1)
5) *Max:* $Max(input1, input2)$
6) *Min:* $Min(input1, input2$
7) *Ephemeral:* $Random.nextDouble$
8) *Func1:* $input1/(1 + input2/parameter)$
9) *func2:* $(input1 * input2)\%255$
10) *func3:* $(input1 + input2)\%255$
11) *func4:* if $input1 > input2$ then $input2$ else $input1$
12) *func5:* $255 - input1$
13) *func6:* $abs(cos(input1) * 255)$
14) *func7:* $(\tan(((input1\%45) * \pi)/180.0) * 255)$
15) *func8:* $abs(tan(input1) * 255)\%255$
16) *func9:* $sqrt((input1 - parameter) * (input1 - parameter) + (input2 - parameter) * (input2 - parameter))$ if > 255 then 255

17) *func10:* $input1\%(parameter + 1) + (255 - parameter)$
18) *func11:* $(input1 + input2)/2$
19) *func12:* if $input1 > input2)$ then $255 * ((input2 + 1)/(input1 + 1))$ else $255 * ((input1 + 1)/(input2 + 1))$
20) *func13:* $abs(sqrt((input1 * input1) - (parameter * parameter) + (input2 * input@) - (parameter * parameter))\%255)$

21) *func14:* $input1 \mid input2$
22) *func15:* $input1$ & $input2$
23) *func16:* $Sin(input)$
24) *func17:* $Cos(input)$
25) *X:* terminal node representing the x coordinate for the texture space
26) *Y:* terminal node representing the y coordinate for the texture space

The terminals used in this language are derived from the X and Y coordinates of the texture space. The bitmap of red, blue, and green are put on top of this texture space without subdividing the area anymore. For example, a 256 by 256 texture space would not become a 25600 by 25600 bitmap. This ensures a simplistic program and one that works with most general purpose images(non-general purpose images will be addressed later). The usage of X and Y terminals also allows up-scaling the final image once the processing is completed. If the processed image bitmaps are only 256 by 256, when outputting the image with the best fitness, it can be scaled to 4000 by 4000 resolution simply by indexing it to 4000 and inserting the index value into the red, blue, and green formulas generated.
The functions used were initially only the basic functions used in the tutorials and for assignment 1. They are functions 1 to 7 listed above. As the experimentation went on, there was clearly some limitations to the potency of these functions. As a result, new functions were introduced in phases into the GP language. The first phase introduced Sine and Cosine in an attempt to capture circular/wavy shapes within figure 1 and figure 3. The second phase tried to improve the overall accuracy by introducing a set of functions mentioned in a research paper cited in the previous section; functions 8 to 22. These phases also inspired benchmarks for comparative experiments later on.

D. Fitness Evaluation

Before the investigation can begin, a fitness function should be defined to measure how ”good” an individual from the population can perform. In the case of this experiment, the difference between the color of each pixel between the image generated and the target image will be the metric of evaluation. This difference can be measured using the true euclidean distance of their red, blue and green color values. It is calculated as such:
$eucliddist = \sqrt{RedDist^2 + GreenDist^2 + BlueDist^2}$
The euclidean distance is often used in mathematics to determine true distance between multi-vector coordinates. This translates seamlessly well since each pixel holds a color that can be represented in a tuple of three.
The total euclidean distance is then calculated by summing up all the euclidean distance of each pixel in the image. The

larger the value of the total distance, the poorer the generated image matches the target image. However, the smaller the value of the total distance, the closer it resembles the target image.

While this euclidean distance works perfectly in math, in procedural textures, it is not all-encompassing. When an image is generated, it is possible that some parts of the image are considered more important than the others. A focal point such as the face of the Mona Lisa for example. That is why if the image has a strong focal point, a mask put on top of the it. The mask works by covering a set coordinate range(usually somewhere in the middle) and any euclidean distance calculated there will be weighted more heavily than the others. This is to penalize the individual for getting an important part wrong more similar to how the euclidean distance punishes large distances more because it is squaring the difference. Taking inspiration from this, the mask that was used was squaring the euclidean distance to severe punish large errors in image focal points. Also, multiplying the distance by ten seems to work just as well without severely damaging the pixels outside the mask. $eucliddist = eucliddist^2$ or $eucliddist = 10 * eucliddist$

```
Mask Pseudo-code:
Get the euclidean distance of the current pixel
If x is between 85 and 170 and y is between 85 and 170
Multiply the euclidean distance by ten
Euclidean Distance Pseudo-code:
RedDist is the difference between the red value of generated and target
BlueDist is the difference between the blue value of generated and target
GreenDist is the difference between the green value of generated and target
Euclidean distance for a pixel is the square root of the sum of squares between RedDist, BlueDist, GreenDist
Total euclidean distance is the sum of all euclidean distance for each pixel. The fitness minimizes this value.
```

III. Results

Results are separated into three sections, namely by experiment 1, experiment 2, and experiment 3. Each experiment has three comparative phases done with different sets of functions. The first phase, phase 0, runs the GP with only the basic functions, 1-7 from the GP Language section. The next phase runs the GP with the basic set of functions from phase 0 but adds the additional cos and sine functions to see if any improvement had been made. Lastly, the final phase(no phase number) uses all previously mentioned functions with the addition of functions 8-22 from the GP language section, which were taken from Ashmore and Miller's research paper titled Evolutionary Art with Cartesian Genetic Programming. A disclaimer that only relatively distinct unique images will be shown in phase 0 and 1 to avoid redundancy. For the final phase, only the best individual is shown followed by the original target image.

A. *Experiment 1*

1) *Phase 0:*

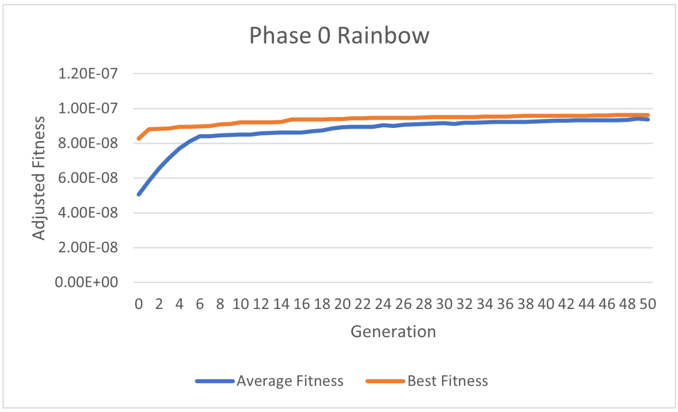

Fig. 4: Phase 0 Rainbow Fitness of 10 Runs

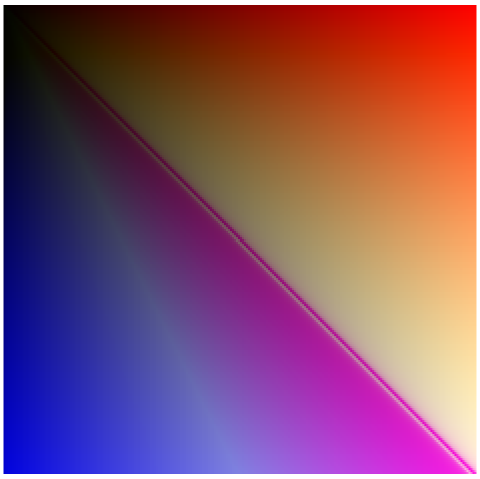

Fig. 5: Phase 0 Unique Rainbow 1

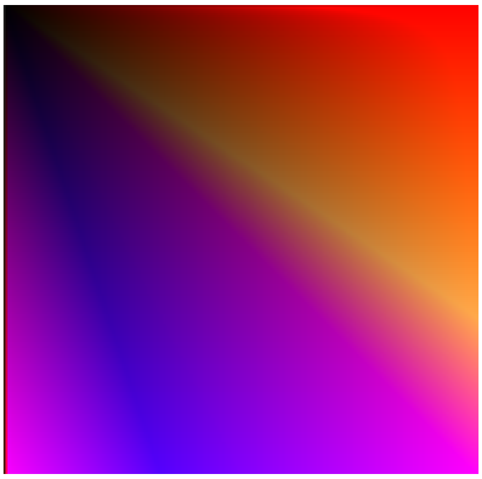

Fig. 6: Phase 0 Unique Rainbow 2

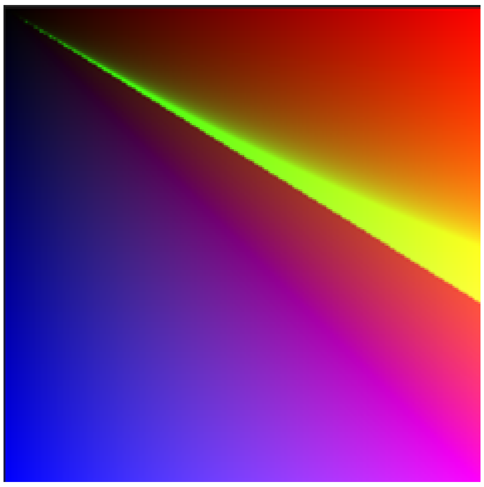

Fig. 7: Phase 0 Unique Rainbow 3

2) *Phase 1:*

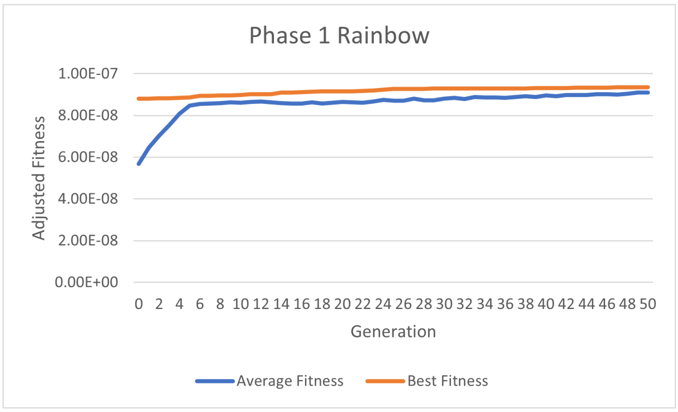

Fig. 8: Phase 1 Rainbow Fitness of 10 Runs

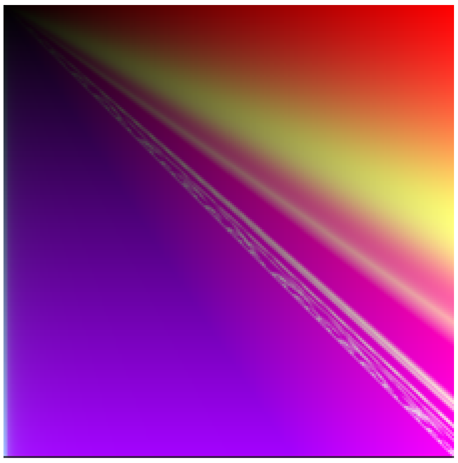

Fig. 9: Phase 1 Unique Rainbow 1

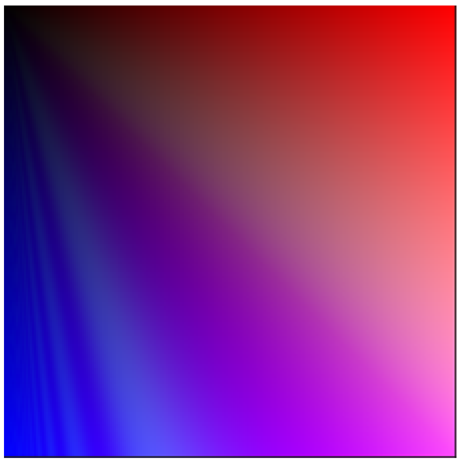
Fig. 10: Phase 1 Unique Rainbow 2

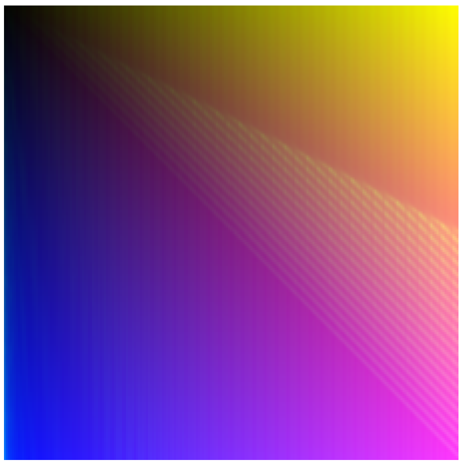
Fig. 11: Phase 1 Unique Rainbow 3

3) *Phase 2:*

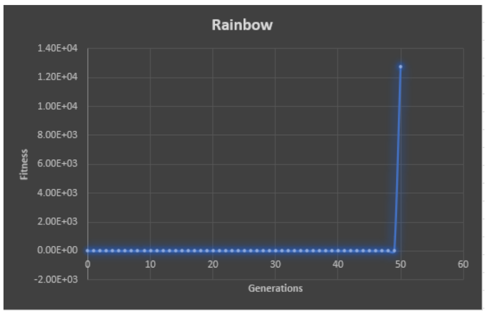

Fig. 12

	Results
Final Fitness	2.2991597033689116E-7
Min	1.03332E-07
Max	2.29699E-07
Mean	1.75975E-07
Median	1.79806E-07
Standard Deviation	3.63712E-08

TABLE II: Figure 12 Table

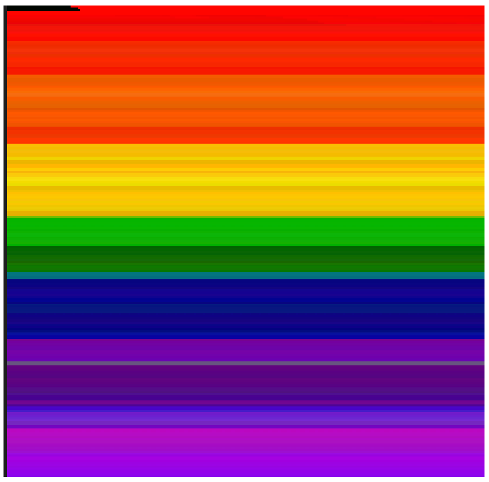
Fig. 13: Gp image of Rainbow

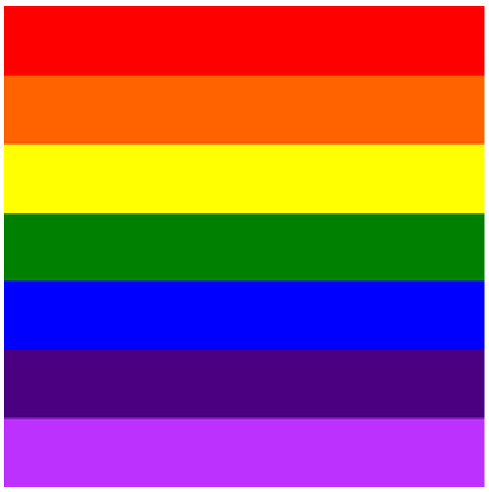
Fig. 14: Target: Rainbow

B. *Experiment 2*

1) *Phase 0:*

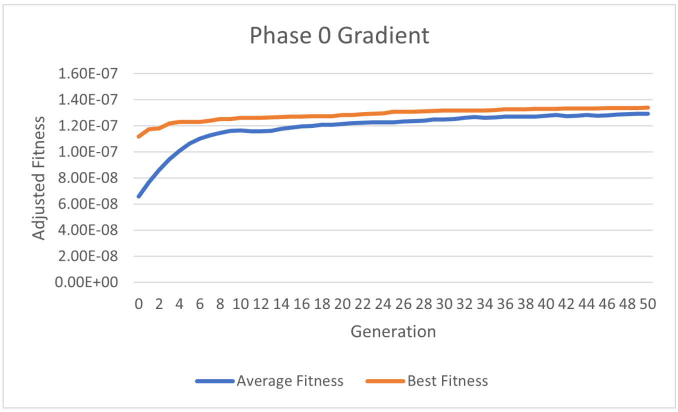

Fig. 15: Phase 0 Gradient Fitness of 10 Runs

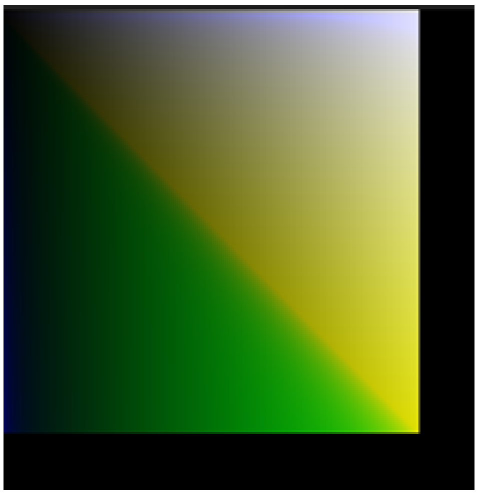

Fig. 16: Phase 0 Unique Gradient 1

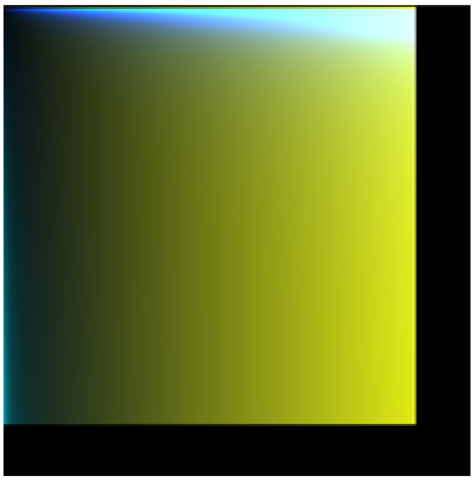

Fig. 17: Phase 0 Unique Gradient 2

2) *Phase 1:*

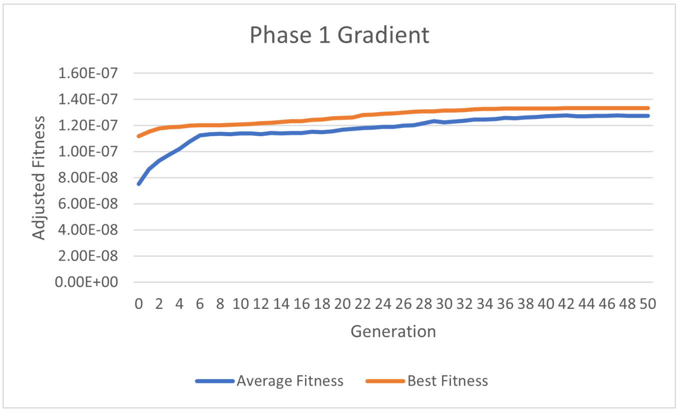

Fig. 18: Phase 1 Gradient Fitness of 10 Runs

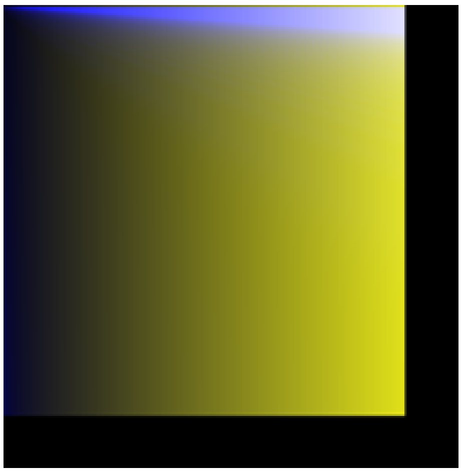

Fig. 19: Phase 1 Unique Gradient 1

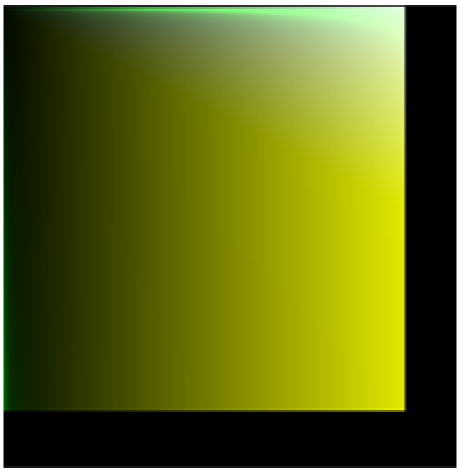

Fig. 20: Phase 1 Unique Gradient 2

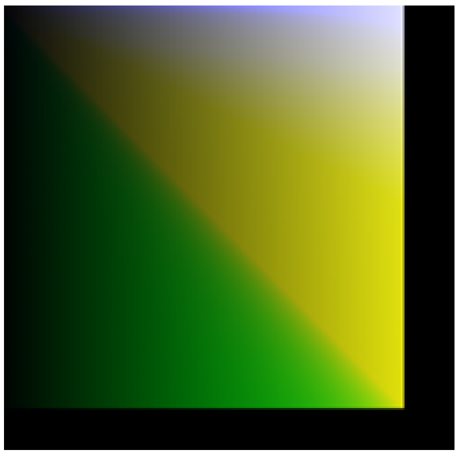
Fig. 21: Phase 1 Unique Gradient 3

3) *Phase 2:*

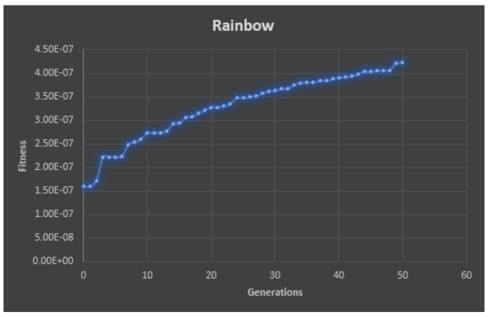

Fig. 22

	Results
Final Fitness	4.2245E-07
Min	9.28205E-08
Max	4.2245E-07
Mean	3.23986E-07
Median	3.47429E-07
Standard Deviation	7.7929E-08

TABLE III: Figure 22 Table

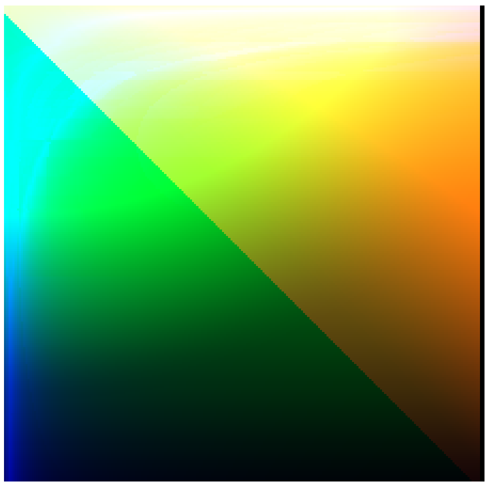

Fig. 23: Gp image of pic1

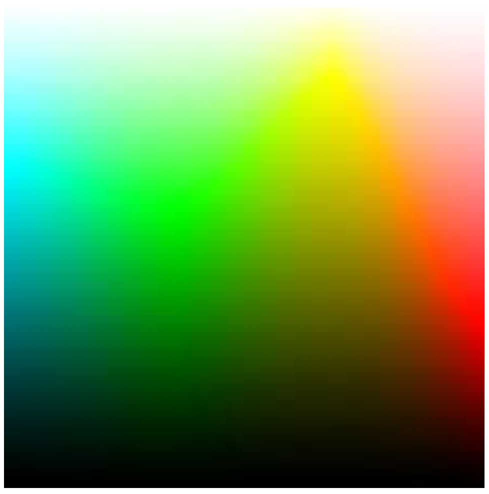

Fig. 24: Target: pic1

C. Experiment 3

1) Phase 0:

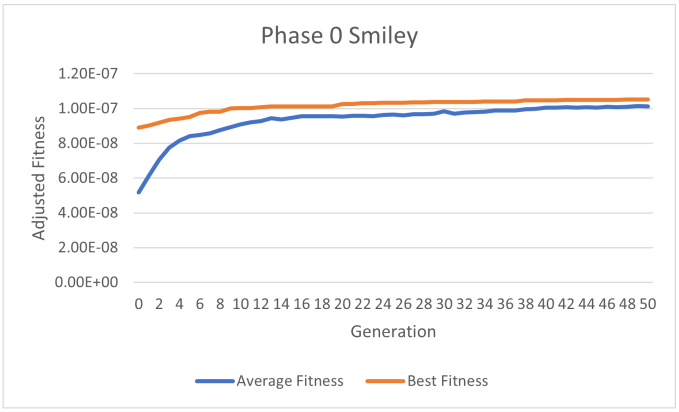

Fig. 25: Phase 0 Smiley Fitness of 10 Runs

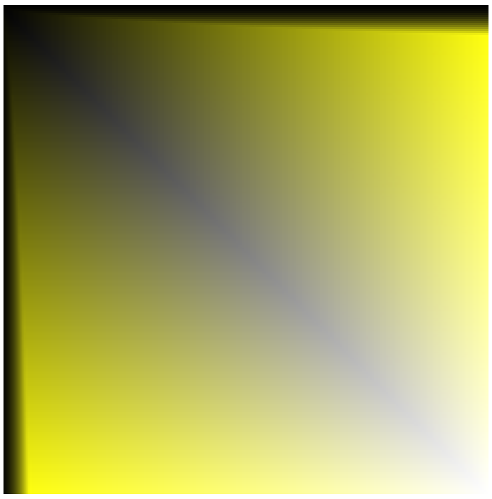

Fig. 26: Phase 0 Unique Smiley 1

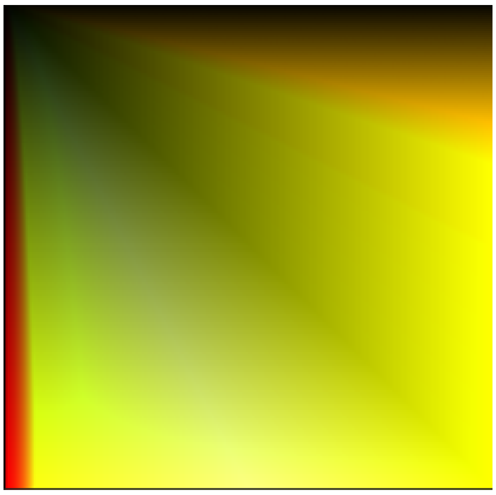

Fig. 27: Phase 0 Unique Smiley 2

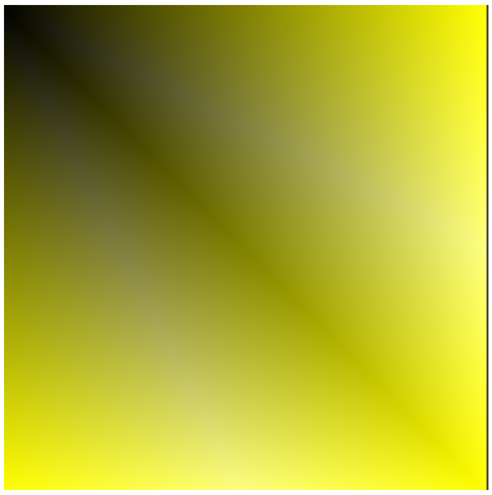

Fig. 28: Phase 0 Unique Smiley 3

2) Phase 1 Maskless:

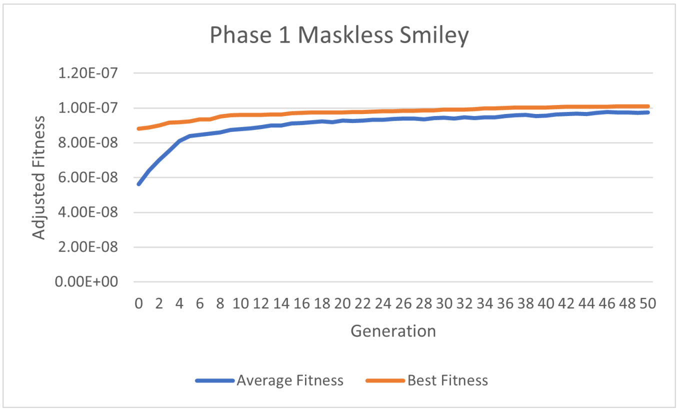

Fig. 29: Phase 1 Smiley Fitness of 10 Runs

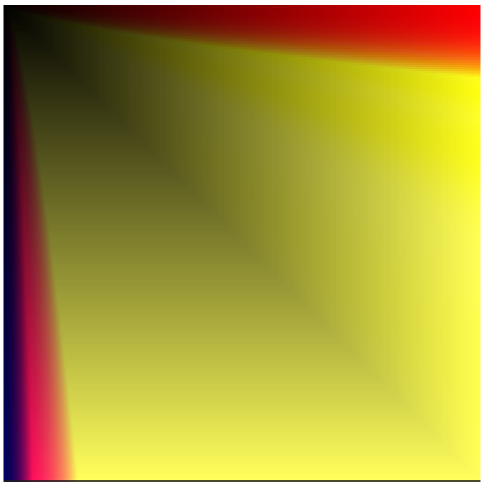

Fig. 30: Phase 1 Unique Smiley 1

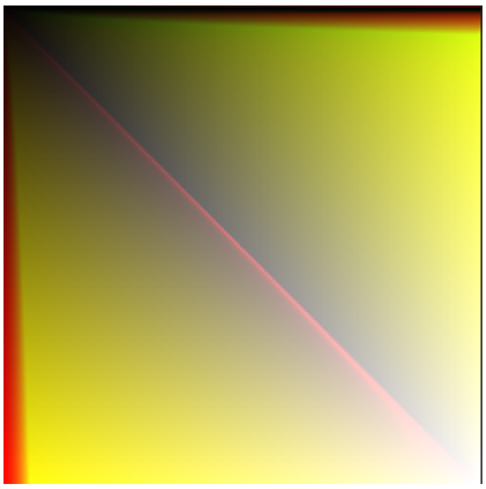
Fig. 31: Phase 1 Unique Smiley 2

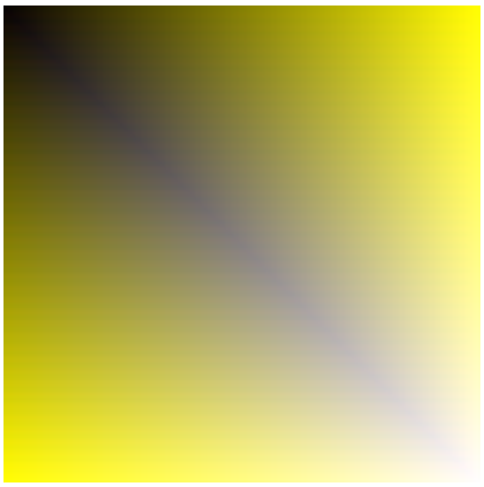
Fig. 32: Phase 1 Unique Smiley 3

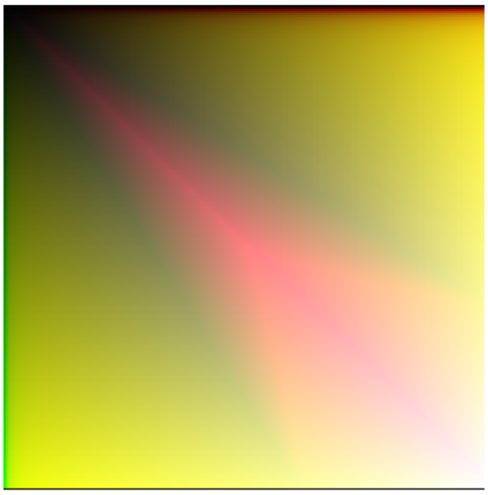
Fig. 34: Phase 1 Unique Smiley 5

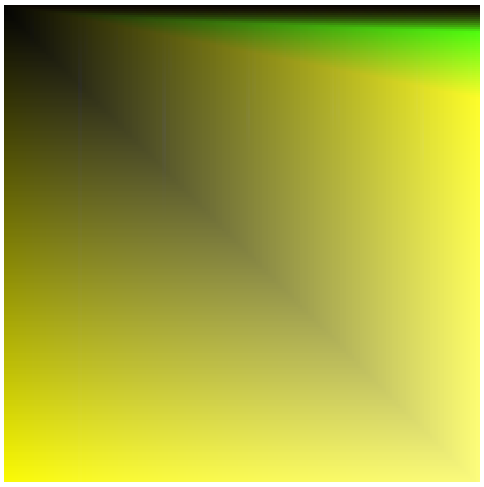
Fig. 33: Phase 1 Unique Smiley 4

3) Phase 1 Masked:

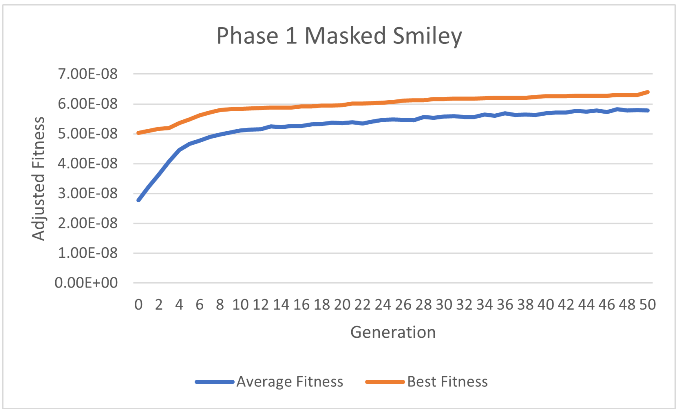

Fig. 35: Phase 1 Masked Smiley Fitness of 10 Runs

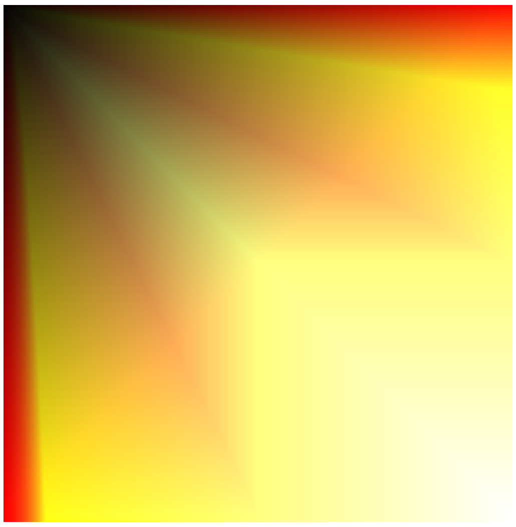

Fig. 36: Phase 1 Masked Unique Smiley 1

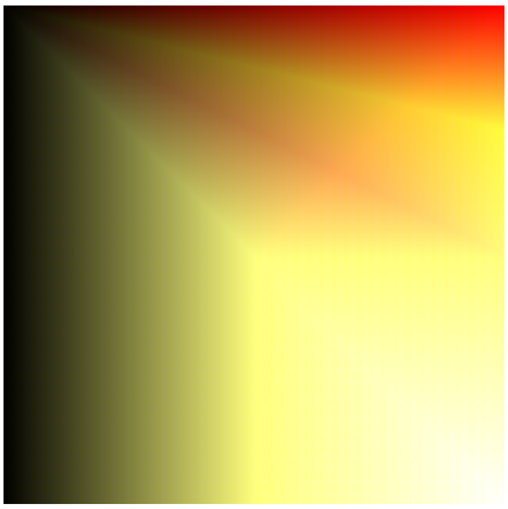

Fig. 37: Phase 1 Masked Unique Smiley 2

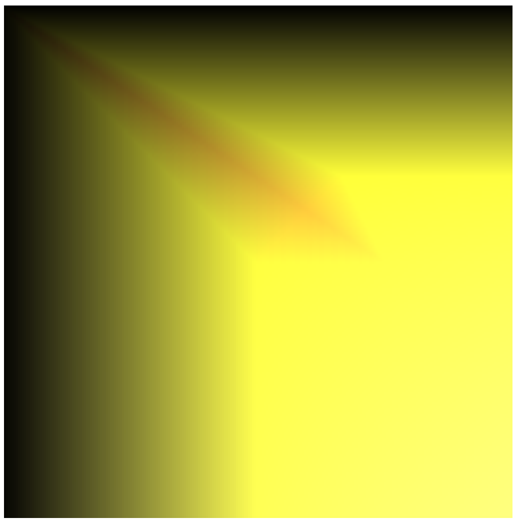

Fig. 38: Phase 1 Masked Unique Smiley 3

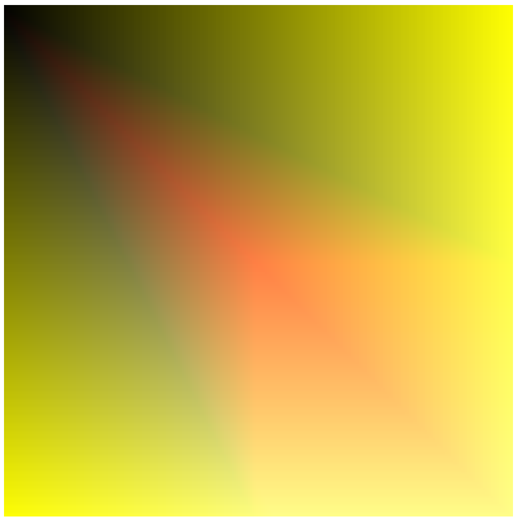

Fig. 39: Phase 1 Masked Unique Smiley 4

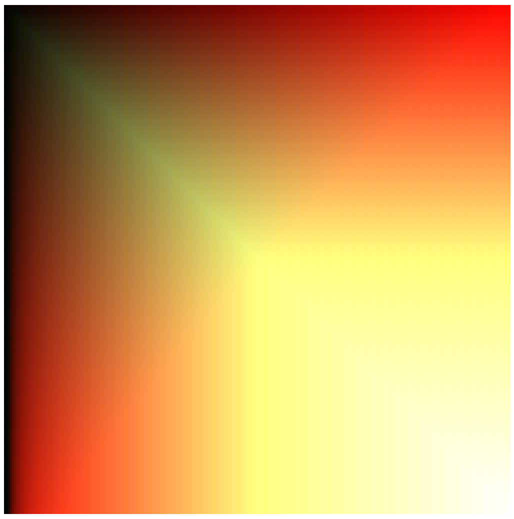

Fig. 40: Phase 1 Masked Unique Smiley 5

4) Phase 2:

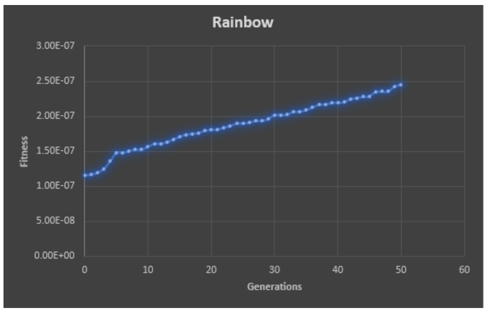

Fig. 41

	Results
Final Fitness	2.455128663299879E-7
Min	1.15618E-07
Max	2.42682E-07
Mean	1.86447E-07
Median	1.89763E-07
Standard Deviation	3.39229E-08

TABLE IV: Figure 41 Table

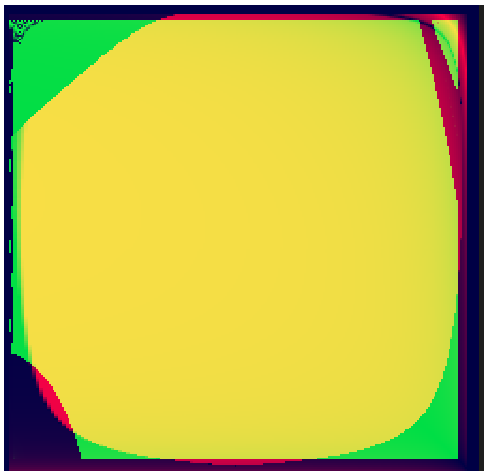

Fig. 42: Gp image of Smiley face

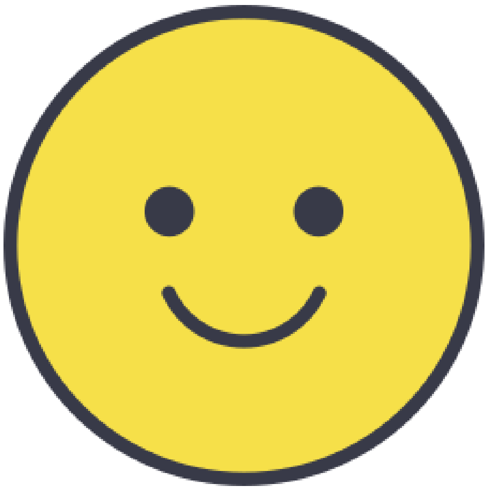

Fig. 43: Target: Smiley Face

IV. Discussion and Comparison

For this experiment, the comparative analysis focused on changing the GP language, i.e. the functions. This was decided after a qualitative investigation of comparing changes in the GP parameters against changes in the GP language. It was readily apparent that the GP language change affected the generated image more so than changing the GP parameters. Intuitively, this also makes sense because changing the GP language allows the system to capture more abstract shapes, color schemes, and increases overall expressiveness.

In experiment 1, figures 5 to 7 were done with very simplistic, mostly linear type functions. In figure 4, the fitness of this experiment tended to significantly improve at the beginning of the run. There were very minor improvements beyond this point. Along with the visual results, it seems to suggest that the improvement was mostly done by finding the color scheme of the original image. In regards to the shape and orientation of the colors, the generated images still seem to be inaccurate. The program is interpreting the rainbow streams as diagonals.

Moving on to phase 1 of experiment 1, The Sine and Cosine functions are introduced into the GP language generating images similar to figure 9 to 11. The fitness measured in figure 8 actually decreased overall compared to phase 0. This is most likely due to the target image not having any round shapes that can be beneficially captured by the oscillating patterns from the Sine and Cosine functions. In fact, the oscillation is disturbing the uniform shading in each unique color block of the rainbow. While some runs ignore the Sine and Cosine for their fittest individual, some individuals with these functions make it out of their run as the fittest such as figure 9. Strips of green oscillations can be visually seen there. These somewhat rare individuals somewhat drags down the overall fitness over 10 runs.

For experiment 1 phase 2, when using rainbow as the target image, the fitness hardly sees any improvements from generation 1 to generation 49, as seen in figure 12. The fitness stays very close to the minimum (see table II) with minuscule improvements. Only at generation 50 does the fitness see a great improvement, jumping to the max/final fitness. By using the additional functions from Evolutionary Art with Cartesian Genetic Programming, the gp saw a qualitative improvement when compared to experiment 1's phase 0 and phase 1.The gp almost recreated the target image with the colouring in at its appropriate coordinates, but with the shading being incorrect in some areas. When looking at figure 13, the image recreated by the gp, and figure 14, the target image, it is evident that that the gp can't distinguish the shading of the same colour, for example, dark blue and light blue or dark green and light green, etc.

To further corroborate the previously mentioned limitation, the next experiment, experiment 2, directly tackles this issue. The target image used in this experiment is an image that blends multiple colours together. The shading of the colour starts off strong then gets lighter, eventually shifting into a different colour, see figure 24.
In phase 0 of experiment 2, using the most basic functions, the images generated were dull and linear as shown in figures 16 and 17. Looking at the fitness in figure 15, rapid improvements can be seen at the beginning of the run but slows down as the run progresses. It is important to note that the growth does not taper off until much later compared to what happened in the phases during experiment 1. The fitness growth during middle of the run could be attributed to the system trying to match the basic orientation of colors. The generated image not only gets the lighter yellow shades and black shades correct but also places them in the correct texture space unlike experiment 1. This could explain a larger growth in fitness during the middle of the run. However, the intricate blend of gradients were not able to be captured using basic functions.
Introducing phase 1 of experiment 2, where the Sine and Cosine functions were inserted once again to generate similar images such as figure 19, 20, and 21. Visually, and statistically, referring to figure 18, there have been minimal changes to the images generated and their overall fitness. Simply put, the new functions introduced do not assist in capturing the details that phase 0 left out. There are little to no round shapes or oscillations present in the target image for the new functions to take advantage of. It can be argued that there may be some humps in the shades of black while examining it closely, but humps are generally parabolic and can be caught using X^2 or $X * X$ variants using the multiplication function in phase 0. Phase 1 therefore had no effect on the overall fitness.
For phase 2 of this experiment, the fitness saw a general upward trend with a few exceptions. For multiple consecutive generations, the fitness stayed the same with no improvements being seen. The first two generations saw no improvement, staying at the minimum fitness, see table 3 and figure 22. After the first three generations, the fitness saw a larger improvement at generation four with smaller improvements succeeding it up until the final generation where it had the best fitness. The results for this phase of the experiment can be seen in figure 23, the gp created image, and 24, the target image. As mentioned in experiment 1 phase 3, the gp is unable is recognize different shadings of colours. When comparing figures 23 and 24, the colours appear in their appropriate texture space coordinated but are unable to transition from dark to light. Another problem that was discovered during this experiment was recreating complicated shapes. In experiment 1 phase 2, the gp was able to recreate rectangles of the corresponding colour, but during this experiment the gp could not recreate the shape in which the colour resided in. The shapes in the recreated image are simple and lacked the complexity of the shapes in the target image.

The final experiment, experiment 3, thus focuses on complexity instead of colour recognition. The target image, figure 43, as a yellow circle with the outline being black, two smaller black circles within the yellow circle, and a black parabola also within the yellow circle, a smiley face.
Phase 0 of experiment 3, had very similar images except their outer border were different as shown in figures 26 to 28. Judging from the fitness function in figure 25, it follows a similar trend in to experiment 2 phases, where the rapid growth at the beginning signifies the general matching of the color palette and the middle mild growth in fitness shows fitting the orientation of the colors. In this case, most of the texture space got colored yellow because that is the only predominant color in the target image. Working from the results generated in figures 26 to 28, the goal is to try to obtain the target figure's shape and/or facial details.
Moving onto phase 1, Sine and Cosine functions are introduced to generate figures 30 to 34. Most of them look similar to the ones in phase 0 but they seem to have inherited some sort of unique diagonal patterns. However, judging from the fitness function in figure 29, this diagonal pattern has no effect whatsoever on the fitness of the generated image. But theoretically, the round and arcing parts of the target image's face should be captured by the Sine and Cosine functions. Unsatisfied with the results, a part B for phase 1 was introduced. The idea to add a mask around the face of the target. Since, the number of pixels in the thin face shape is low as well as the small texture area of the face details, it was possible that those pixel colors were forsaken for the rest of the pixels in the image. Therefore, the facial pixels should be given more importance by weighing them accordingly. In the case of this experiment, the facial pixels weighted ten times more than any of the other pixels. The figures 36 to 40 are the unique results generated. The figure 35 gives a general understanding to the fitness but it cannot be used to compare to the other fitness due to the scaling of weights offsetting the fitness value. Regardless of this, the main pattern of a rapid fitness growth in the beginning to match color palettes, then a mild middle growth to match the orientation is still present within the curve. Defeated, the masking ended in failure and

the analysis moves on towards its final phase.
Phase 2's fitness after the first few generations saw a linear upward trend with a very small standard deviation, see figure 41 and table 4. The fitness increased in a stable manner with no outliers as it arrived at the best fitness at generation 50. The results of this phase concluded that the gp cannot recreate complicated images as seen in figure 42. The gp had gotten the majority of the colours correct, which was as expected, but when it can to recreating the image, it got the general shape of the yellow circle correct but failed in all other aspects. In order to recreate the image, the performance of the colour recognition in the appropriate texture space saw a decrease in accuracy as seen when comparing figure 42 and 43, the colour black cannot be seen within the yellow circle. Thus when trying to recreated complex images, the GP sees a decrease a qualitative decrease in its performance.

V. Hardware and Software Limitations

There have been some issues that were restricting the success of this experiment. First and foremost, the runs themselves take a long time to complete. This led to limiting the number of runs for each experiment. Typically, a number of 30 runs would be statistically viable but 10 runs were done instead due to the time it takes per run and the given time allotted for this assignment.
The solutions from these experiment were taken from a machine running ECJ on a virtual machine and also on the sandcastle server. By using a virtual machine, there were some complications with allowing the virtual machine to use the GPU to accelerate the process. It was way too time consuming and is worth a project by itself. Similarly, running ECJ on the sandcastle server also had its limitations. While threading was allowed, the school server limited the number of threads that could be use. Furthermore, threading a single experiment requires each thread to have their own time seed. Our familiarity with ECJ does not allow this to be done error free. Instead of threading a single experiment, another option would be making each thread do its own run. Again, our familiarity with ECJ is not at the level that the output functions can be manipulated to store and label all runs without overwriting one another or other potential complications.
ECJ had another problem while running on the sandcastle server. The server by default had a CPU time limit which terminates the running program after a certain amount of time. At the time, the maximum population size and maximum generation ECJ could handle without timing out was 300 and 30 respectively. There happened to be a work around this issue by typing in 'ulimit -t unlimited' in the command line. While it no longer stopped the program before it could converge, every now and then the server would disconnect the user.
There is another matter regarding non-general purposed images. Normally when an image is read in, it usually has a 256 by 256 resolution at the minimum; usually more nowadays with high resolution images. However, if the original image is under the set minimum resolution, it would not render properly into the program. Also, even if the image meets the minimum resolution requirement, if a small portion of the image is set as a target for the GP system to recreate such as the eyes or nose of the Mona Lisa, it would also not render properly. The bitmap was not designed to subdivide the small texture space of X, Y into suitable I,J bitmaps of red, blue, and green. For example, if Mona Lisa's nose was located between -1.0 and 1.0 for X and Y, the bitmap would not create 100 by 100 I,J bitmaps. Therefore, for this experiment, a non-general purpose image is defined by a low resolution original image or a small focal point of a normal resolution image. They are not covered by the software.

VI. Conclusion

This paper compared the results of different gp languages on an evo-art system. Three experiments with their own three sub-experiments were performed on images that had varying difficulty. The first experiment was performed with a very simple image, rainbow. Phase 0 of this experiment used basic gp languages as stated before in this paper and the results were poor. The gp image looked nothing like the target image and didn't have all the colours the target image had. The next phase, phase 1, performed better then phase 0, having most of the colours in the image but still looked nothing like the target image. Phase 2 performed the best, recreating an image that looked almost like the target image. It could not exactly match the target because it had a difficult time recognizing shading. Where light blue was in the target image, the gp had dark blue or where dark green was required, it had a mix of both. To recognize the limitation of this system, the next image for experiment 2 was designed to focus on this problem. Once again, phase 0 and phase 1 performed the worst when trying to recreate the image and phase 2 performed the best. Even though phase 2 performed the best, the gp created image was anything but. It was able to recognize the colours but could not accurately recognize shading. In this experiment, another problem was discovered with the system, it could not recreate complicated shapes, which experiment 3 was used to corroborate. This experiment used the target image of a smiley face. Phase 0 and phase 1 were able to accurately produce an image containing the colours of the smiley face but were unable to recreate the shape. Whereas phase 2 got relatively close to creating a circle but the only colour it accurately outputted was yellow. Overall, when needing to recreate a simple image, it is best to use the gp languages that were used in phase 2, but in order to recreate complicated images, more sophisticated languages will be needed to tackle the problem.

VII. Bibliography

Smiley Face. (n.d.). icons8. Retrieved March 18, 2022, from https://icons8.com/illustrations/smiley-face.

Gando, F. (n.d.). Rainbow Theme. Tooliphone. Retrieved March 18, 2022, from https://iskin.tooliphone.net/ios-theme-without-jailbreak-theme-8558-user-12969.

Gradient of Colors. (n.d.). gstatic. Retrieved March 18, 2022, from https://encrypted-tbn0.gstatic.com/images

Miller, J. F. (2011). Introduction to evolutionary computation and Genetic Programming. Cartesian Genetic Programming, 1-16. https://doi.org/10.1007/978-3-642-17310-3_1

Luke, S. (n.d.). ECJ/ECJ at master · GMUEClab/ECJ. GitHub. Retrieved March 20, 2022, from https://github.com/GMUEClab/ecj/tree/master/ecj

Ibakurov. (n.d.). REGRESSIONPROBLEMONGPWITHECJ/regressionproblem/SRC at master · Ibakurov/REGRESSIONPROBLEMONGPWITHECJ. GitHub. Retrieved March 20, 2022, from https://github.com/ibakurov/RegressionProblemOnGPWithECJ/tree/master/RegressionProblem/src

Grantham, K. (n.d.). Assignment 2 parameters. Computer science. Retrieved March 20, 2022, from https://www.cosc.brocku.ca/Offerings/5P71/assignments//ass2.params

VIII. Appendix

A. *Rainbow-Tree*

Tree 0: (func4 (func5 (func15 (func8 1.0) (func13 (func11 (Max (func14 1.0 (func13 1.0 Y)) 1.0) Y) Y))) (Max (func15 (func15 (func8 1.0) (Max (func13 Y (func15 (func8 1.0) (func13 (func11 (Max (func14 1.0 Y) 1.0) Y) Y))) (func13 (Max (func13 (Max (func14 1.0 Y) (func1 (Max (func13 (Max (func14 1.0 Y) (func1 X Y)) (func15 (func8 1.0) (func13 1.0 Y))) (func1 X Y)) Y)) (func15 (func8 1.0) (func13 1.0 Y))) (func1 X Y)) (func11 (Max Y 1.0) Y)))) (Max (func13 Y 1.0) (func13 (Max (func13 (Max (func13 Y 1.0) (func1 X Y)) (Max (func14 1.0 Y) (func1 X Y))) (func1 X (func11 (func8 1.0) (Max (func1 Y X) Y)))) (func15 (func8 1.0) (func13 1.0 Y))))) (func13 (Max (func14 1.0 Y) (func1 X Y)) Y)))

Tree 1: (func9 (func15 (func15 (func13 Y (func15 (func13 Y (func13 Y Y)) (- 1.0 Y))) (- 1.0 Y)) (- 1.0 Y)) (func15 (func13 Y (func13 Y Y)) (- 1.0 Y)))

Tree 2: (func15 (func9 Y Y) (func15 (func11 Y (- (func15 Y (func15 (func15 (func15 Y (func11 Y Y)) (func9 (func8 1.0) Y)) (func15 Y (func11 Y (func11 Y (func2 Y (/ (func8 1.0) (func10 X)))))))) (func1 Y (* (func11 Y X) (func4 1.0 Y))))) Y))

B. *pic1-Tree*

Tree 0:
(func14 (func14 (func1 (func9 (func14 (func1 (func14 (func5 Y) (func5 Y)) (func5 Y)) (func2 (- X Y) 1.0)) (- X Y)) (- X Y)) (func1 (func2 (- X Y) 1.0) (- X Y))) (func9 (func14 (func1 (- X Y) (- X Y)) (func9 (func9 (func14 (func1 (func2 (- X Y) 1.0) (- X Y)) (func5 Y)) (- X Y)) (- X Y))) (- X Y)))

Tree 1:
(func9 (func9 (func9 (func9 (func9 (func9 (func9 (func9 (func5 Y) (func5 (func9 Y (func9 Y (func9 Y (func9 Y X)))))) (func5 (func9 Y (func9 Y X)))) (Max 1.0 1.0)) (func5 (func9 Y (func9 (func9 Y X) (func5 Y))))) (Max 1.0 1.0)) (func5 (func9 Y (func9 (func9 Y (func5 Y)) X)))) (func5 (func9 Y (func9 Y X)))) (func5 (func9 Y (func9 Y X))))

Tree 2:
(func12 (func1 (func12 (func9 1.0 1.0) (/ Y (func7 (func1 (/ (func12 (func9 1.0 1.0) (/ (/ X 1.0) 1.0)) 1.0) (func14 (/ Y (func7

1.0)) (/ Y (func12 (func9 1.0 1.0) (/ X 1.0))))))) (func14 (/ Y 1.0) (+ (func12 (func1 (func12 (func9 1.0 1.0) (/ Y (func7 1.0))) (func14 (/ Y 1.0) (+ (func12 (func9 1.0 1.0) (func9 1.0 1.0)) Y))) (/ Y (func12 (func9 (func9 1.0 1.0) 1.0) (/ X 1.0)))) Y))) (/ Y (func12 (func9 1.0 1.0) (/ X 1.0))))

C. Smiley Face-Tree

Tree 0: (func4 (func4 (* (func11 (func4 (func4 (* (func11 (func4 (func4 (* (+ (func11 (func9 Y Y) (- (+ (func13 Y 1.0) (func13 X 1.0)) (func5 Y))) (func13 X 1.0)) Y) (func12 1.0 (func5 Y))) (func12 (func1 (func10 (func13 (func13 Y 1.0) 1.0)) Y) (func5 Y))) (- (func5 (func13 X 1.0)) (func5 (func1 Y Y)))) Y) (func12 1.0 (func5 Y))) (func12 (func1 (func10 (func13 (func13 Y 1.0) 1.0)) Y) (func5 Y))) (- (func14 1.0 X) (func5 (func1 Y Y)))) Y) (func12 1.0 (func5 Y))) (func12 (func1 (func10 (Max (func4 (func4 (* (+ (func11 (func9 Y Y) (- (func14 1.0 X) (func5 Y))) (func13 X (func7 Y))) Y) (func12 1.0 (func5 Y))) (func12 (func1 (func10 (func13 (func5 (func13 X 1.0)) 1.0)) Y) (func5 Y))) (func8 X))) Y) (func5 (func13 X 1.0))))

Tree 1: (func12 (func12 (func6 (* (Max X (func13 (func11 1.0 Y) (/ Y Y))) Y)) (func13 (func10 1.0) (/ Y Y))) (func13 (* (* (func14 (func13 (func3 Y 1.0) (/ (func13 1.0 (func5 Y)) (func13 X 1.0))) (Max X X)) (func12 (func6 (* (Max (func12 (func6 (/ Y Y)) (func13 (func3 Y 1.0) (/ (func13 1.0 (func5 Y)) (func13 X 1.0)))) (func6 (* (Max X (func13 (func11 1.0 (func5 (Max X X))) (/ (func11 1.0 Y) Y))) Y))) Y)) (func5 1.0))) (func13 (- (* Y Y) (func9 (func10 (* (func4 (func6 (func13 (func11 1.0 (func5 Y)) (/ Y Y))) (func12 (func12 1.0 Y) (func12 (func6 (* (Max X (func13 (func11 1.0 Y) (/ Y Y))) Y)) (func13 (func3 Y 1.0) (/ (func13 1.0 (func5 Y)) (func13 X 1.0)))))) Y)) (func2 X Y))) (func13 1.0 X))) (/ Y (func13 1.0 (func13 (func13 (func11 1.0 (func5 Y)) (/ Y Y)) X)))))

Tree 2: (func11 1.0 (func6 1.0))

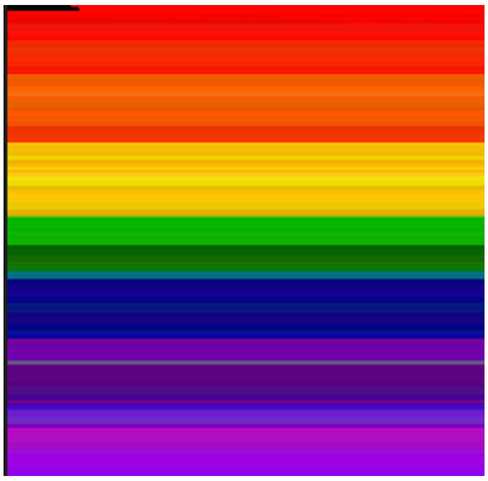
Fig. 44: Large output: 500x500

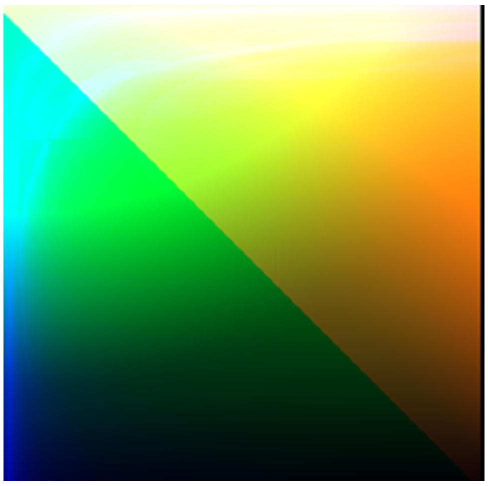
Fig. 45: Large output: 500x500

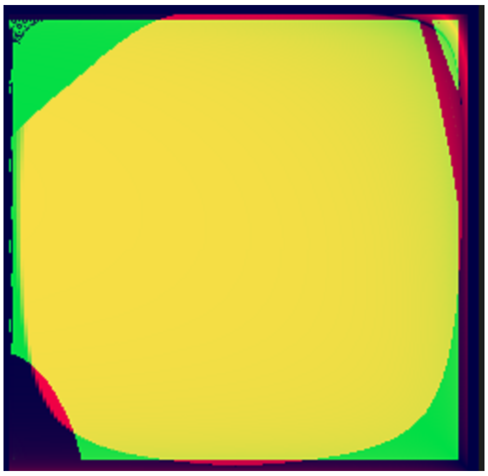
Fig. 46: Large output: 500x500