Marshall Jones

Final Project: Post Analysis

Computer Science 209

Description

Background

The Picasso Project is an application that allows the user to create expressions, which are evaluated into colors, then into images. The final product can create a beautiful array of colorful images.

Purpose

The purpose of this project was to develop experience working together as a team on complex objectives, and to refine skills in code strategy, design, and product development. The intended user might be for a student of mathematics who wants to visualize the output of mathematical expressions in colors, or a talented graphic designer who takes a scientific approach with their designs. With enough care, the images produced by this application could be used on their own or as a background in conjunction with other designs to create logos, graphics, and even for non-fungible token (NFT) projects, to name a few use cases.

Extensions

My team implemented three extensions: a View Assigned Button, View Multiple Images, and a Random Equation Generator.
The View Assigned Button extension allows the user to view a collection of assigned expressions and their identifiers, along with a small pixmap of the given expression. This extension updates every time a new window is opened.
The View Multiple Images extension allows the user to view evaluated expressions in a separate window when they click the "view in new Window" button. The user can view different expressions in as many windows as they want.
The Random Equation Generator extension allows the user to randomly generate a new expression by pressing a button in the GUI. The new expression is then displayed on the screen.

Planning

At the very beginning, understanding all the pieces of the project was overwhelming. I felt like I needed to put 6 hours of reading just to understand all the moving parts. After breaking the project into its logical components, however, I found that it wasn’t as complicated as it seemed. I thought the extensions were the most ambiguous part of the project, while the Picasso language functions and operators were very clear. Thus, the functions and operators were the easiest to implement for me.

Status

The ExpressionTreeNode is the most abstract form of the expression tree; it implements the EvaluatableExpression interface so that ExpressionTreeNode and its child classes can evaluate into an image. The child classes of ExpressionTreeNode include UnaryFunction, MultiArgumentFunction, UnaryOperator, and BinaryOperator. Each of these child classes are abstract classes intended to reduce repetitive code in our many function and operator classes. Thanks to this design, it was quite straightforward to implement the functions and operators once we had the abstract classes implemented. This design was my favorite part of the project's makeup.

Testing went very well. We divided tests into ErrorParsedEvaluatorTests, EvaluatorTests, ParsedExpressionTreeTests, a ParserTestDriver, and TokenizerTests. The EvaluatorTests file was very long, and if I had more time I would have separated it into separate logical files that utilize @BeforeEach to initialize the common functionalities.

Code Analysis Conclusions

Our tests required the most design rework, yet our functions and operators required the most code rework. This was interesting and surprising to me – we did a great job with the design and implementation of the functions' and operators' parent classes.

In many cases, the evaluator tests helped us to understand what was wrong with our operators and functions. In some cases, like with multi-argument functions, our tests just weren't written correctly and needed to be redesigned. But for the most part, the tests helped us find errors that we would not have otherwise been able to find. For example, using the escape character in the configuration file for the negate operator, and redesigning the division and modulus functionality so that the program doesn't fail when dividing by zero.

When we ran the program and it failed on an expression, it was most time-consuming when we hadn't written tests for that functionality yet. We quickly found that the best way to debug these faults was by writing test cases that cover those expressions. This allowed us to quickly identify the error(s) and address them.
The most frequent faults, however, were the ones found in unit testing. They were also the easiest to address.
For future projects, I would say that writing test cases should absolutely be prioritized, even though it takes a significant amount of time. The pain of finding errors that hadn't been tested for tended to compound as other parts of the program were impacted by those errors. That pain greatly outweighed the minimal pain of finding an error during testing.

Collaboration

The Picasso Eternals were an outstanding team. We were very productive and collaborative. I encountered extenuating circumstances that prevented me from getting anything done in time for the first deadline. My teammates were very understanding. They picked up my slack and handled my assigned part.

Collaborating with my team was for the most part easy. We used GroupMe as our communications channel, but sometimes it was difficult to get in contact with each other in a timely manner. We ended up having a lot of calls over the phone and Zoom as we collaborated on error handling. Git was helpful for collaboration and coordination, and we also used a product called Asana at the beginning of our project. Asana is a project management tool like a

Kanban. I think it would be very useful for GitHub to incorporate a similar tool with communication functionality into its interface.

Armando: 30%

- GUI
- Extension
- Operators
- Functions
- Tests
- Debugging
- Managing

Mesoma: 25%

- Extension
- Operators
- Functions
- Tests

Nobel: 25%

- Extension
- Operators
- Functions
- Tests
- Debugging

Marshall: 20%

- Operators
- Functions
- Tests
- Debugging
- Managing

I would definitely say that everyone carried their weight throughout the project. We spread the work relatively evenly, and everyone was willing to fulfill the roles they were assigned. We did a great job collaborating and spreading tasks across the board so that everyone was able to develop a strong understanding of the project as a whole. Having a good understanding of the workings of the whole application was important because it allowed us to help each other when one person was stuck or error handling.

Future Work and Future Students

I wish we had extra time to add more extensions. I think it would have been very cool to design a random expression generator which then generates variations of that expression as a moving picture, like a screensaver.

It would also be interesting to add more functionality with buttons that allow someone to fine-tune an expression in terms of colors, hue, vibrance, shape, etc.

Finally, it would be very useful to add a few base expressions that produce standard shapes that a graphic designer would use frequently.