LambdaGators: Lambda Calculus Illustrator

By Anarav Patel, Charles Rule, and Michael Rizzo

The goal was to make an application that can dynamically illustrate the computational steps of lambda calculus with the alligator analogy Professor Ames taught us in class. We loved this analogy as much as he did; it demystified lambda calculus by turning it into a fun subject. We wanted to take it one step further by making it into an interactive version that goes beyond basic expressions. Hopefully, future students will learn lambda calculus from our final project and have the same enjoyable experience.

We initially wanted to create a webapp using Heroku, Flask, and Python. As we were working on making and loading images, we decided the rendering of the alligators and eggs would be easier with a game engine. We settled on Pygame as our engine of choice, and set to work on making a desktop application instead of a web application.

The first challenge we faced was making a lambda calculus interpreter that outputs the steps of the reduction. This was done using three tools: a lexer that tokenized the original input, a parser that determined which part of lambda calculus a token belonged to, and an interpreter that performed beta reduction, substitution and (greedy) alpha conversion. The interpreter works iteratively and stores each iteration in an array. These steps are then displayed in the GUI. Additionally, we implemented an Abstract Syntax Tree (AST) for storing the tokens and traversing them while interpreting. Because the parser adds parentheses to the input string as a way to identify applications when necessary, we eliminated 'dead' alligators from the original metaphor, since it is not possible for the interpreter to distinguish a 'dead' alligator from an 'alive' one.

The second challenge was a collection of GUI problems; namely, coloring and generating the alligators. There is no uniform color change function in Pygame, so we had to make sure the sprites were in a base color that worked with Pygame's transform function. Additionally, we chose to render a fixed amount of alligators and eggs so the application would run smoothly. This limits the user to 26 single lowercase letter names for functions or variables. This is enough for the average user to test a wide variety of functions.

The final challenge was linking these two aspects so that when a lambda expression is interpreted, the correct collection of alligators and eggs shows up. Initially, setting up the GUI to have basic buttons and a text box was difficult, as Pygame doesn't support it out-of-the-box. We looked into GUI libraries, but we could not find any that would gel together with Pygame. Since our GUI elements were going to be limited anyway, we decided to add GUI elements manually. For the rendering of the cursor, we used an implementation from online (which has been cited appropriately) and modified that code for our needs. Another challenge was rendering the alligators and the eggs in an efficient manner. Initially, we had a version working that would dynamically generate and update the renderings, but we had to revise our implementation because it was burdening our computer resources and had frame rate issues. Since we were learning a lot of the concepts as we went, there might be issues at times with rendering, but we managed to research and optimize it enough where it doesn't tax the computer resources too much.

Overall, this project was a success, as it has a more in-depth lambda calculus interpreter and more comprehensive alligator graphics than other projects. If we had more time, we would update the parser to catch more edge cases and add animations in between reductions. This was an awesome opportunity to explore lambda calculus, programming language interpretation and Python; specifically, how to develop an application drawing from what we learned in Principles of Programming.