Mini-Project 4

Coleman Ellis and Kaitlyn Keil

March 10, 2016

Abstract

Using object-oriented programming and the Pygame library, we created an interactive circle-plot display, a graphic which draws connections between the digits of an endless decimal number.

1 Results

Our interactive circle-plotter draws a circle with different sections representing single digits, 0 through 9.

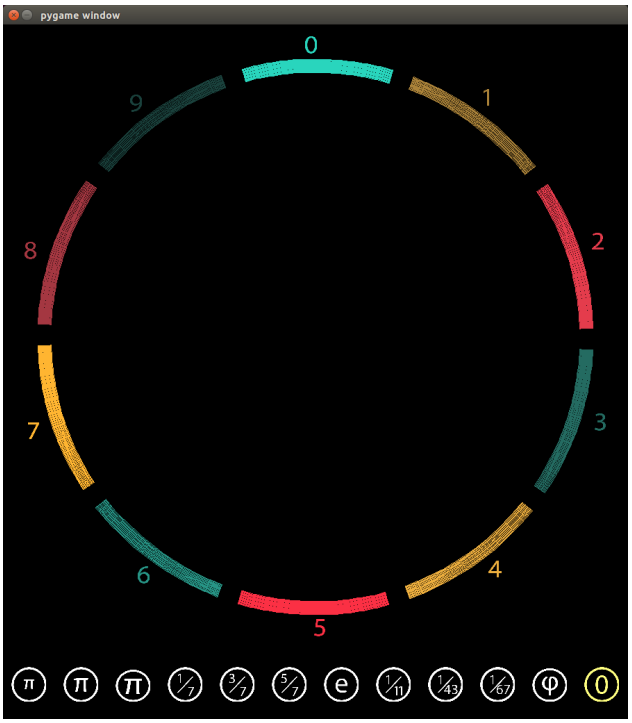

Figure 1: The circle of sections and their corresponding labels before a number has been selected.

Using the buttons below, the user can select different irrational and repeating decimals to see visualized, such as π (the three different buttons yield more digits of π, allowing the viewer to understand how the process works), ϕ, and several fractions.

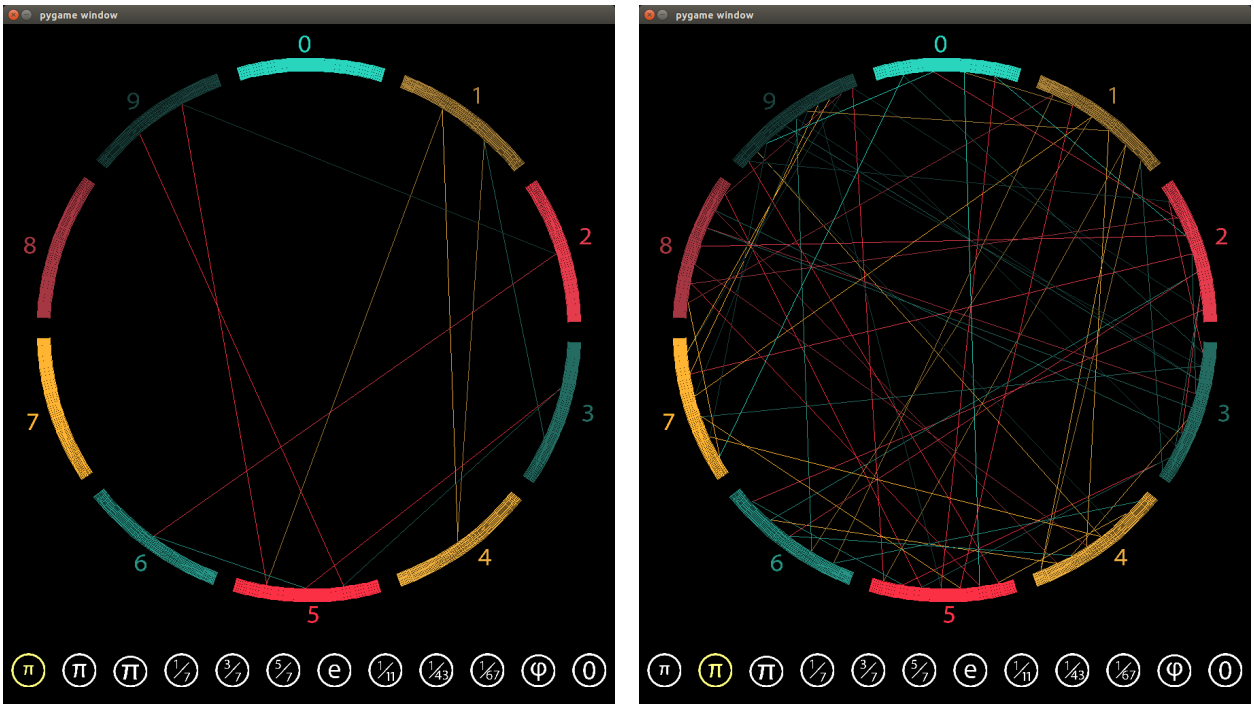

Figure 2: The left screenshot shows the first few digits of π, which makes it easier to trace which number connects to which. On the right is the next step up, beginning to show the complexity.

Each line represents a connection between two adjacent digits. In π, for example, there's a line drawn from 3 to 1, from 1 to 4, from 4 to 1, etc. The color of the line is the same as the color of the section it starts from (the line from 3 to 1 would be the same color as the 3 section).

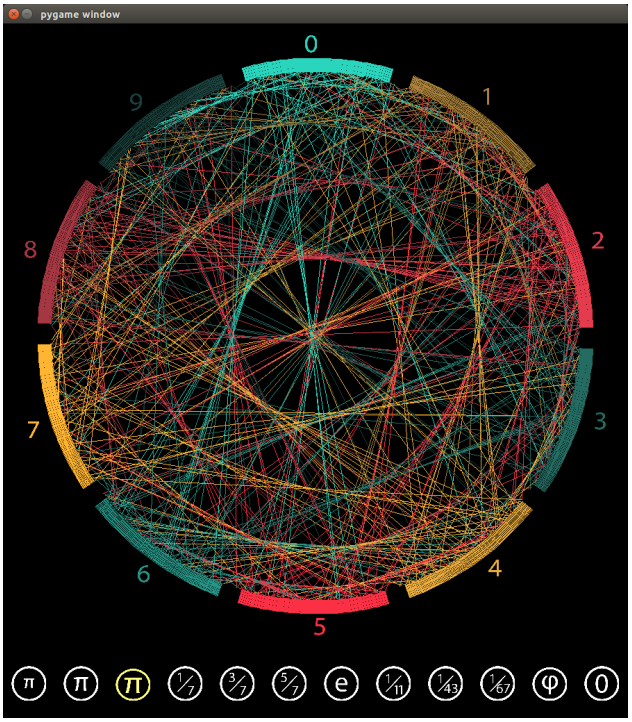

Figure 3: The first 545 digits of π. At this point, the individual connections are difficult to follow, but the graph is very pretty.

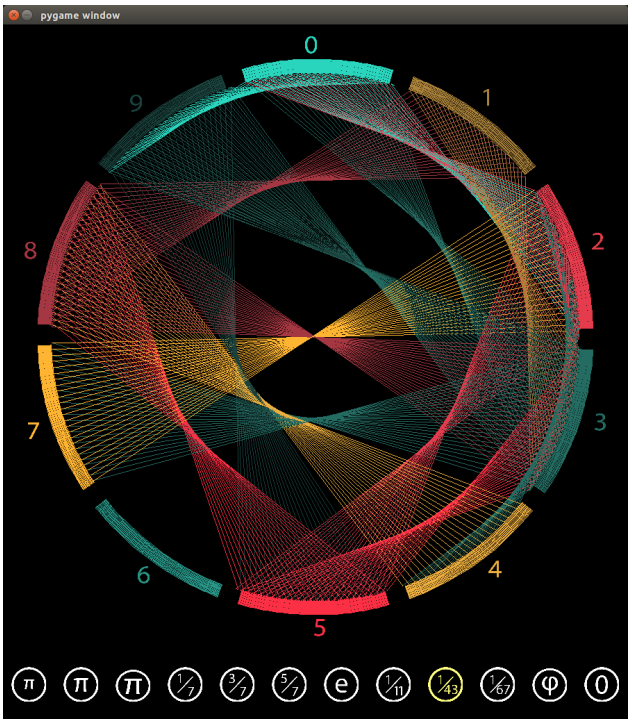

Figure 4: Representation of $\frac{1}{43}$. This is a repeating decimal, as can be seen by the groupings of connections.

2 Implementation

Our program as a whole didn't delve too deep into the inheritance side of object-oriented programming, but we did define and use a fair number of classes. The most-used classes that we defined were the ElementArc (representing the numbered sections) and ConnectionArc (representing the lines between the Element Arcs) classes, which were the core of our visualization, and the CirclePlotModel class, which generates and stores the requisite Element and Connection arcs. The Label class is just a label for a numbered section (with the label imported from an image instead of PyGame-drawn text so the text would look a bit better), the ModelButton class gives us clickable buttons and an easy way to switch between displayed model, and the CirclePlotView draws all of the elements of our model, filling in the View of the Model-View-Controller trinity. The Controller element of said trinity exists in the loop in our program's main function, where we watch for mouse events, check if any of the buttons are clicked and, if they are, switch to the model that corresponds to the clicked button.

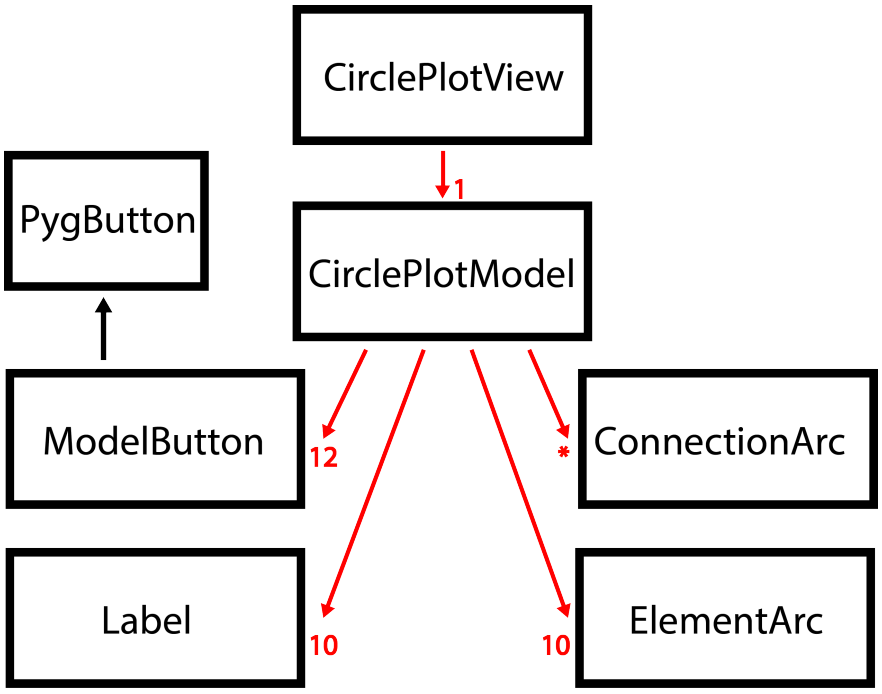

Figure 5: The UML class diagram of our project, with red arrows indicating 'has a' relationships and the black arrow showing an 'is a' connection. We chose to exclude pygame classes (such as Surface, Rect, and Color) for simplicity's sake, and due to the fact they would all be 'has a' connections.

The one instance where we did use inheritance was for ModelButton, which inherits from PygButton (a library we found on the PyGame website). This choice allowed us to avoid creating interactive buttons from scratch, as PygButton contains functionality for creating, displaying, and handling events of buttons. This was partly due to time and partly due to a desire not to reinvent the wheel. ModelButton, the inheriting class we created, allowed us to easily pass in and recolor an image to serve as the visual of the button, well as stored the number needed to generate the button's corresponding Plot Model.

3 Reflection

Our base project goal was fairly well scoped for the time available to us. Both of us were busy enough to have some difficulty meeting to work on this project, but we managed to budget the time we did have appropriately to produce most of the features we wanted, though we didn't make our stretch goal. Our unit testing also worked quite well, even if we couldn't use doctests for the majority of the code. We started with making the sections, then drew a couple of connections, added labels, added multiple options for diagrams, and finally made the buttons to be able to switch between them.

Though it feels as though we did not use objects and inheritance very much, we used them appropriately for our project. Having the minimum number of objects to complete our goal helped with readability and still let us learn the basics of object-based programming, both with and without inheritance. While it might have been better to choose a slightly more object-based project, we still both grasped the concepts of objects, inheritance, and the Model-View-Controller trinity.

When we began, we intended to do mostly pair-programming. Overall, we did in fact follow through on that, except for a few instances when Coleman was able to stay up and work later than Kaitlyn. Coleman took most of the image creation for labels and buttons, and Kaitlyn found PygButton as an option for the user-interaction. We worked together well, and neither of us felt as though the other did less work.