**Comprehensive Final Project Report:**

**Simple Microsoft Paint Application**

Aditya Verma, 002342095

Submitted to Professor Christopher Diggins for CS 321

April 26 2023

**INTRODUCTION**

My C# project ‘SimplePainterApplication’ realises a simple version of the popular Microsoft Paint utilizing the many features of the SVG (Scalable Vector Graphics) rendering library under the NuGet package manager. My project is a user-friendly painting application designed to enable users to draw various shapes, free-draw, and save their work in different formats. The application also enables users to perform actions such as clear, delete, zoom-in, zoom-out, etc. This report discusses the application’s features and its robust event-handling in detail, highlighting any known bugs in the project along with the struggles during debugging processes.

**FUNCTIONALITY**

**Shape Drawing/Free Draw:**

The application supports multiple shape types, including standard shapes like *Ellipse, Rectangle, Square,* and *Circle*. Additionally, the application supports drawing more complex shapes like the *Star, Cloud,* and *CloudToEllipse*. Users can select a shape type and draw it on the canvas by clicking and dragging the mouse. The application also allows users to set the stroke colour, fill colour, and stroke thickness for each shape. To make the drawing experience more aesthetically pleasing, I enabled a dotted-line preview of the shape being drawn as the user drags the shape across the canvas. This allows the user to preview the shape getting drawn before the user chooses to finalise its desired size.

**Clear Canvas:**

The clear functionality enables users to remove all shapes and free-drawn drawings from the canvas and start with a blank canvas. This feature comes in handy when the user wishes to begin another drawing without having to close the application and reopen the canvas.

**Save Options:**

The application offers multiple ways to save the users work at their desired location. Users can save their drawings as image files in various formats, such as PNG, JPEG, and BMP.

**Select Capability using Point-Containment Check:**

The Contains() method in the Shape class allows users to check whether a given point is within the bounds of a shape. This feature is essential for implementing the shape selection functionalities in the application.

**Special Shape Drawing Implementation:** Clouds, Star and Cloud to Ellipse

* *DrawStar()*

By iterating through ‘numpoints’, using the ‘step’ and ‘angle’ variables, the position of each point in the star is calculated. The function keeps track of the distance from the centre to the outer and inner points of the star. The function then connects these points using the ‘pen’, which forms the star.

* *DrawCloud()*

Since I already had functionality to draw circles, I realised I could use multiple smaller circles to form a cloud. First calculating the cloudRadius, the function calculated the radius of each circle depending on the size of the cloud desired. Then it keeps track of all the centres of each circle. Using 6 circles total, arranged in a specific pattern, the function then creates a cloud-like appearance.

* *Cloud to Ellipse*

While implementing the aforementioned special shapes, I thought it would be fun to try and implement a shape that drags as a cloud but draws as a circle when the mouse is finally lifted. It was a fun experiment and I was able to include this functionality in my final version of the project by assigning a cloud during the ‘mouseMove’ event handler, and updating it to a circle in the ‘mouseUp’ event handler.

**Keyboard Shortcut Functionalities for Buttons:**

For the toolbox elements that do not allow setting shortcut keys directly in their properties, I use the ‘Form\_KeyDown’ keyboard event handler that uses conditionals to detect certain keyboard combinations to allow for keyboard shortcuts for associated actions. The shortcuts implemented in my project are displayed in the following table:

|  |  |
| --- | --- |
| **Action** | **Shortcut** |
| *Using ‘Form\_KeyDown’* | |
| Zoom-In | Ctrl + Up |
| Zoom-Out | Ctrl + Down |
| *Using Properties* | |
| Save | Ctrl + S |
| SaveAs | Ctrl + Shift + S |
| Exit | Ctrl + Q |
| Import SVG | Ctrl + I |
| Export SVG | Ctrl + E |
| Clear | Ctrl + C |
| Undo | Ctrl + Z |

**ADVANCED FEATURES**

**Import SVG/Load SVG and Export SVG:**

The application enables users to export their work as an SVG file at their desired location. Additionally, the application offers users to import any SVG file on their system into the application and display it on the canvas.   
  
**Zoom Functionality:** Zoom-In/Zoom-Out

I was particularly interested in learning how to implement the zoom functionality as zooming is a feature often used while using painting applications like MS Paint. My project enables users to zoom in and out of the canvas to work on finer details or view the entire drawing. This functionality supports keyboard shortcuts. The project utilizes an ApplyZoom() method that implements a fixed zoom factor, manually modifiable, which is applied to the canvas to determine the strength of the zoom.

**KNOWN BUGS AND ISSUES**

**Shape Selection and Manipulation**

I was able to implement ability for users to select a shape by clicking on it. My project highlights the selected shape with a red dotted-line boundary to indicate shape selection. However, I struggled with using the ‘\_actions’ List of Shapes to enable manipulation of the selected shape. On double-clicking the selected shape, I attempted to write a function ‘picturebox1\_DoubleClick’ that would delete the shape on the canvas. I think the reason why I was not able to implement this functionality was because I could not invoke deletion in the ‘\_actions’ List of Shapes successfully.

**Project Abstraction**

I would have liked to delegate the various functionalities based on models such as the selection model responsible for shape selection and manipulation, drawing model dealing with all drawing abilities, canvas model integrating zooming functionality, etc. Because I focussed more on learning about how to use the SVG library for my features, I skipped on trying to use abstraction in my project implementation. I ensured I followed the C# coding guidelines, however, if I were to redo my project, I would use better class organization and delegated models to make my project even more coherent to a third-party viewer.

**CONCLUSION**

The initial plan was to include a Lasso selection ability. This, I learned was not supported by the SVG rendering library. I would need an independent library to attempt to do this. I tried using Windows Forms’ in-built framework, but found it to be too complex. I dropped the plan after numerous failed attempts and went with a click-to-select functionality instead using SVG. I’m glad I did this as I could explore more of what the SVG library had to offer.

Overall, working on this simple painter application was a challenging yet highly enriching journey. I explored numerous features of a new rendering library, SVG, that I used to implement some basic functionalities of the famous Microsoft Paint in my project. It was enlightening to relearn that developing a robust application takes time, as it is a product of rigorous case-testing. I was able to practice how to efficiently debug errors and work from the simplest test up in complexity, to define a robust implementation for my desired features. I thank Professor Diggins for introducing the wonders of C# and for a memorable winter semester.