GCC Code Coverage Report

Directory: ./
File: App.cpp
Date: 2021-11-17 01:36:45

	Exec	Total	Coverage
Lines:	162	172	94.2%
Branches:	68	124	54.8%

```
Line  Branch   Exec  Source
   1                 /**
   2                  *  @file   App.cpp
   3                  *  @brief  Main class for program
   4                  *  @author Mike and Dennis Ping
   5                  *  @date   2021-11-13
   6                  ************************************************/
   7
   8                 // Include our Third-Party SFML header
   9                 #include <SFML/Graphics.hpp>
  10                 #include <SFML/Graphics/Image.hpp>
  11                 #include <SFML/Graphics/Texture.hpp>
  12                 #include <SFML/Graphics/Sprite.hpp>
  13                 #include <SFML/Window.hpp>
  14                 // Include standard library C++ libraries.
  15                 #include <cassert>
  16                 #include <iostream>
  17                 #include <string>
  18                 #include <queue>
  19                 // Project header files
  20                 #include "App.hpp"
  21                 #include "Draw.hpp"
  22                 #include "MathUtility.hpp"
  23
  24                 /*! \brief  App constructor
  25                 *
  26                 */
  27                39  App::App() {
  28                         // Canvas variables
  29                13      m_window = nullptr;
  30    ▶ 2/4       13      m_image = new sf::Image;
  31    ▶ 2/4       13      m_sprite = new sf::Sprite;
  32    ▶ 2/4       13      m_texture = new sf::Texture;
  33    ▶ 2/4       13      m_current_color = new sf::Color;
  34                13      m_paintbrush_radius = nullptr;
  35    ▶ 2/4       13      m_cursor_sprite = new sf::Sprite;
  36    ▶ 2/4       13      m_cursor_texture = new sf::RenderTexture;
  37    ▶ 2/4       13      m_cursor_circle = new sf::CircleShape(5);
  38    ▶ 1/2       13      m_circle_template = new std::vector<std::pair<int,int>>;
  39
  40                         // Color code member variable
  41   ▶ 9/18      117      color_codes = {
  42                13          {sf::Keyboard::Num1, sf::Color::Black},
  43                13          {sf::Keyboard::Num2, sf::Color::White},
  44                13          {sf::Keyboard::Num3, sf::Color::Red},
  45                13          {sf::Keyboard::Num4, sf::Color::Green},
  46                13          {sf::Keyboard::Num5, sf::Color::Blue},
  47                13          {sf::Keyboard::Num6, sf::Color::Yellow},
  48                13          {sf::Keyboard::Num7, sf::Color::Magenta},
  49                13          {sf::Keyboard::Num8, sf::Color::Cyan}
  50                         };
  51                26  }
  52
  53                26  App::~App(){}
  54
  55                     /*! \brief  Clear the redo stack.
  56                     *
  57                     */
  58              5010  void App::ClearRedo() {
  59              5010      int clearCount = 0;
  60    ▶ 2/2     5107      while(!m_redo.empty()) {
  61                97          clearCount++;
  62                97          m_redo.pop();
  63                         }
  64    ▶ 2/2     5011      while (!m_redo_count.empty()) {
  65                 1          m_redo_count.pop();
  66                         }
  67    ▶ 2/2     5010      if (clearCount > 0) {
  68                 1          std::cout << "Cleared " << clearCount << " from the redo stack" << std::endl;
  69                 1      }
  70              5023  }
  71
  72                     /*! \brief      Add new commands to queue for execution.
  73                     *
  74                     */
  75              5010  void App::AddCommand(std::unique_ptr<Command> c) {
  76              5010      m_commands.push(std::move(c));
```

```
            5010  }
                  
                  /*! \brief  Execute commands from the m_command stack.
                  *           Push executed commands to the undo stack.
                  *           Clear the redo stack.
                  */
            1130  int App::ExecuteCommand() {
            1130      int successCount = 0;
 ▶ 2/2      6140      while (!m_commands.empty()) {
            5010          bool success = m_commands.front() -> execute();
 ▶ 2/2      5010          if (success) {
            1633              m_undo.push(std::move(m_commands.front()));
            1633              successCount++;
            1633          }
            5010          m_commands.pop();
            5010          ClearRedo();
                      }
            1130      return successCount;
                  }
                  
                  /*! \brief  Look at the m_undo_count stack to determine how many Draw commands
                  *           to undo. The opposite logic of the RedoCommand().
                  */
               5  int App::UndoCommand() {
                      // Need this if statement so we don't undo "nothing"
               5      int numUndo = 0;
 ▶ 2/2         5      if (!m_undo_count.empty()) {
               3          numUndo = m_undo_count.top();
               3          std::cout << "Undoing: " << m_undo_count.top() << " pixels" << std::endl;
 ▶ 2/2       294          for (int i = 0; i < m_undo_count.top(); i++) {
             291              m_undo.top() -> undo();
             291              m_redo.push(std::move(m_undo.top()));
             291              m_undo.pop();
             291          }
               3          m_redo_count.push(m_undo_count.top());
               3          m_undo_count.pop();
               3      }
                      else {
               2          std::cout << "There is nothing to undo" << std::endl;
                      }
               5      return numUndo;
                  }
                  
                  /*! \brief  Look at the m_redo_count stack to determine how many Draw commands
                  *           to redo. The opposite logic of the UndoCommand().
                  */
               3  int App::RedoCommand() {
                      // Need this if statement so we don't redo "nothing"
               3      int numRedo = 0;
 ▶ 2/2         3      if (!m_redo_count.empty()) {
               1          numRedo = m_redo_count.top();
               1          std::cout << "Redoing: " << m_redo_count.top() << " pixels" << std::endl;
 ▶ 2/2        98          for (int i = 0; i < m_redo_count.top(); i++) {
              97              m_redo.top() -> redo();
              97              m_undo.push(std::move(m_redo.top()));
              97              m_redo.pop();
              97          }
               1          m_undo_count.push(m_redo_count.top());
               1          m_redo_count.pop();
               1      }
                      else {
               2          std::cout << "There is nothing to redo" << std::endl;
                      }
               3      return numRedo;
                  }
                  
                  /*! \brief  Return a reference to our m_image, so that
                  *           we do not have to publicly expose it.
                  *
                  */
          230633  sf::Image& App::GetImage(){
          230633      return *m_image;
                  }
                  
                  /*! \brief  Return a reference to our m_Texture so that
                  *           we do not have to publicly expose it.
                  *
                  */
               7  sf::Texture& App::GetTexture(){
               7      return *m_texture;
                  }
                  
                  /*! \brief  Return a reference to our m_window
                  *
                  */
              20  sf::RenderWindow& App::GetWindow(){
```

```
           20      return *m_window;
                }

                /*! \brief  Return a reference to our m_sprite
                *
                */
           10   sf::Sprite& App::GetSprite(){
           10       return *m_sprite;
                }

                /*! \brief  Return a reference to our m_current_color
                */
        10410   sf::Color& App::GetPaintbrushColor() {
        10410       return *m_current_color;
                }

                /*! \brief  Set the m_current_color to the newColor
                *
                */
           14   void App::SetPaintbrushColor(sf::Keyboard::Key numKey) {
           14       *m_current_color = color_codes[numKey];
           14   }

                /*! \brief  Return a reference to our m_paintbrush_radius
                *
                */
            7   int& App::GetPaintbrushRadius(){
            7       return *m_paintbrush_radius;
                }

                /*! \brief  Set the m_paintbrush_radius to the new radius.
                *
                */
            4   void App::SetPaintbrushRadius(int radius){
            4       *m_paintbrush_radius = radius;
            4   }

                /*! \brief  Set the sprite cursor position on the window and apply an offset because
                *           the pointer tip is not exactly in the center of the cursor.
                */
           13   void App::SetCursorPosition(const int &x, const int &y) {
           13       m_cursor_sprite->setPosition(x - *m_paintbrush_radius, y - *m_paintbrush_radius);
           13   }

                /*! \brief  Generate a new cursor if the paintbrush radius or color is changed.
                *
                */
           13   void App::GenerateCursor(int radius, sf::Color paintbrush_color) {
           13       radius += 1;
                    // Build our circle shape
           13       m_cursor_circle -> setRadius(radius);
           13       m_cursor_circle -> setFillColor(paintbrush_color);
           13       m_cursor_circle -> setPointCount(4*radius);
▶ 1/4      13       if (paintbrush_color == sf::Color::Black || paintbrush_color == sf::Color::Blue) {
           13           m_cursor_circle -> setOutlineColor(sf::Color::White);
           13       }
                    else {
            ✗           m_cursor_circle -> setOutlineColor(sf::Color::Black);
                    }
           13       m_cursor_circle -> setOutlineThickness(-1);

                    // Create the cursor texture and draw our circle shape on it
           13       m_cursor_texture -> create((radius)*2, (radius)*2);
           13       m_cursor_texture -> clear(sf::Color::Transparent);
                    //m_cursor_texture -> setSmooth(true);
           13       m_cursor_texture -> draw(*m_cursor_circle);
           13       m_cursor_sprite -> setTexture(m_cursor_texture -> getTexture(), true);
           13   }

                /*! \brief  Generate and cache a circle template for drawing.
                *
                */
           15   void App::GenerateCircleTemplate(int radius) {
           15       *m_circle_template = MathUtility::BresenhamCircleAlgo(radius);
           15   }

                /*! \brief  A cached circle template generated at (0,0) which is shifted by (x,y)
                *           rather than constantly computing the same circle.
                */
           18   std::vector<std::pair<int,int>> App::UseCircleTemplate(int x, int y) {
           18           std::vector<std::pair<int,int>> transformedCircle;
                    // Reserve memory in vector to get 4x performance: https://github.com/facontidavide/CPP_Optimizations_Diary/blob/maste
▶ 1/2      18       transformedCircle.reserve((m_circle_template->size()*8));
                    // Use std::transform to shift the circle template by (x,y)
▶ 3/6      18       std::transform(m_circle_template->begin(), m_circle_template->end(), std::back_inserter(transformedCircle),
         2084           [x,y](std::pair<int,int> p) {
          2066                  return std::make_pair(p.first + x, p.second + y);
                            });
            18      return transformedCircle;
▶ 1/2       18  }

                /*! \brief      Destroy all raw pointers before ending the program.
                *
                */
            13  void App::Destroy(){
▶ 1/2       13      delete m_current_color;
▶ 1/2       13      delete m_paintbrush_radius;
▶ 1/2       13      delete m_cursor_sprite;
▶ 1/2       13      delete m_cursor_texture;
▶ 1/2       13      delete m_cursor_circle;
▶ 1/2       13      delete m_circle_template;
▶ 1/2       13      delete m_image;
▶ 1/2       13      delete m_sprite;
▶ 1/2       13      delete m_texture;
▶ 1/2       13      delete m_window;
            13  }

                /*! \brief  Initializes the App and sets up the main
                *           rendering window(i.e. our canvas.)
                */
            13  void App::Init(void (*initFunction)(void)){
                    // Create our window
            13      int width = 600;
            13      int height = 400;
                    // sf::ContextSettings settings;
                    // settings.antialiasingLevel = 16;
▶ 4/10      13      m_window = new sf::RenderWindow(sf::VideoMode(width,height),"Mini-Paint alpha 0.0.3",sf::Style::Titlebar | sf::Style::
            13      m_window -> setVerticalSyncEnabled(true);
                    // Set the mouse cursor to be invisible because we are going to draw our own cursor
            13      m_window->setMouseCursorVisible(false);
                    // Create an image which stores the pixels we will update
            13      m_image->create(width, height, sf::Color::White);
▶ 2/4       26      assert(m_image != nullptr && "m_image != nullptr");
                    // Create a texture which lives in the GPU and will render our image
            13      m_texture->loadFromImage(*m_image);
▶ 2/4       26      assert(m_texture != nullptr && "m_texture != nullptr");
                    // Create a sprite which is the entity that can be textured
            13      m_sprite->setTexture(*m_texture);
▶ 2/4       26      assert(m_sprite != nullptr && "m_sprite != nullptr");
                    // Initialize current color = black
            13      SetPaintbrushColor(sf::Keyboard::Num0);
                    // Initialize the cursor radius
            13      m_paintbrush_radius = new int(5);
                    // Generate the cursor with current color = black
            13      GenerateCursor(*m_paintbrush_radius, sf::Color::Black);
                    // Set the cursor initial position to off screen so it doesn't momentarily appear
            13      int x = -10;
            13      int y = -10;
            13      SetCursorPosition(x, y);
            13      GenerateCircleTemplate(5);
            13      m_initFunc = initFunction;
            13  }

                /*! \brief  Set a callback function which will be called
                *           each iteration of the main loop before drawing.
                *
                */
            10  void App::UpdateCallback(void (*updateFunction)(App& myApp)){
            10      m_updateFunc = updateFunction;
            10  }

                /*! \brief  Set a callback function which will be called
                *           each iteration of the main loop after update.
                *
                */
            10  void App::DrawCallback(void (*drawFunction)(App& myApp)){
            10      m_drawFunc = drawFunction;
            10  }

                /*! \brief      The main loop function which handles initialization
                            and will be executed until the main window is closed.
                            Within the loop function the update and draw callback
                            functions will be called.
                *
                */
             ✗  void App::Loop(App& myApp){
                    // Call the init function
             ✗      m_initFunc();

                    // Start the main rendering loop
             ✗      while(m_window->isOpen()){
                        // Clear the window
```

```
✗          m_window->clear();
           // Updates specified by the user
✗          m_updateFunc(myApp);
           // Additional drawing specified by user
✗          m_drawFunc(myApp);
           // Update the texture
           // Note: This can be done in the 'draw call'
           // Draw to the canvas
✗          m_window->draw(*m_sprite);
✗          m_window->draw(*m_cursor_sprite);
           // Display the canvas
✗          m_window->display();
       }
   }
```

Generated by: GCOVR (Version 5.0)