WebImage

Class Description
The main goal of this class is to create a C++ wrapper for HTML images so that they can be easily implemented into an Emscripten based web interface. This class will be a helpful tool that allows for easier image usage such as handling image loading, positioning, size, visibility and image accessibility (alt text, etc).

Similar Classes
Within the standard library I believe that you should be familiar with std::string, std::optional, and std::tuple to inform the functionality of this class. The string library will be helpful for handling image paths as well as URLs for images, the optional library is helpful to use when there is trouble loading images correctly, and the tuple library is helpful for storing image positioning data. All of these libraries are crucial to consider when looking at how this class should function.

Key functions

The functions for this class should be:

WebImage(const std::string& source)
WebImage(const std::string& source, const std::string& alt_text = “”)
void SetSource(const std::string& source)
void SetAlt(const std::string& alt_text)
std::string GetSource() const
std::string GetAlt() const

std::tuple GetPosition() const
void SetPosition(int x, int y)
void SetX(int x)
void SetY(int y)
int GetX() const
int GetY() const

void SetSize(int width, int height)
void SetWidth(int width)
void SetHeight(int height)

void Show()
void Hide()
bool IsVisible()

Error Conditions
Some error conditions that this class will need to be responsive to are invalid size or position values, failure to load images, image formatting errors, and invalid URL formats. Invalid size or position values would be a programmer error that could be resolved using asserts, image formatting and failure to load images are recoverable errors and can be handled using exceptions, and invalid URL formats are due to user error which can be handled by returning False or another value to indicate and handle the error.

Expected Challenges
Challenges I expect to come across are related to general use of Emscripten for the first time and the best way to implement this class while using it. I also am worried about image loading when integrated with Javascript, and would like to better understand the best way to do this to avoid errors with C++ trying to load synchronously. Other than these issues I can only think of cross browser compatibility and challenges coordinating with other classes and groups.

Related C++ classes of other groups
I expect to interact mostly within my group with the classes WebCanvas and WebLayout. These two classes will manage the loaded images from my class, and use the images within the canvas. Within group 21 I will likely need to align with the ImageManager and ImageGrid classes to ensure that there is cross compatibility within our classes and APIs. I also anticipate interacting with group 13 to coordinate with their point, circle, box, and surface classes.

WebTextbox

Class Description

The class of WebTextbox is a simple class that aims to be able to manage Textboxes in HTML, through the use of C++ code. At its highest level of functionality, the class should be able to clearly and distinctly display relevant information about whichever textbox is being changed. This includes, but is not limited to; Font color, Font family, and Font size, as well as any other information pertaining to the same.

Similar Classes
The classes in standard C++ which relate to this would most likely be string, and ostream, as these are the two most common ways of outputting collections of words onto your screen. The string_view functionality might come into handy when paired with std::string as a sort of “data storage” from which text will be displayed.

Key functions
The functions to be implemented are as follows:
WebTextBox: As a constructor to bind the object to an HTML element by its ID

Void setText(const std::string& text): For the content inside the textbox
Void setColor(int r, int g, int b): For the visible color of the text
Void setFontSize(int size): For setting the size of the text on screen
Bool isValid(): To check if the data is in fact, valid

Error Conditions
Regarding Programmer errors, possibly things such as passing an empty string to the elementID would probably cause problems, and could be fixed by asserting that the ID is not, in fact, empty. Another programmer issue could be something such as inputting an RGB value over 255, and could also be solved with an assert. A potentially recoverable error could be that the elementID provided is not found in the document containing all IDs. This issue could be solved by bypassing exceptions, and instead checking the isValid() indicator and return immediately. The final type of errors are user errors, and could consist of errors like inputting text that breaks HTML's rendering, like "<" or ">". This could also be quickly solved by sanitizing the user's inputted text if it contains any illegal characters.

Expected Challenges
There is almost certainly going to be some overhead with the functions I use, such as standard "cin". This type of a blocking wait could cause some slow-down in the code's overall efficiency, as it halts for user input, before resuming. There is also going to be a lot of memory allocation that I need to correctly implement for the same. Alongside this, security vulnerabilities, such as HTML injections are something I will definitely need to consider.

Related C++ classes of other groups
My most critical C++ class that I need to collaborate with is within my group, and it is WebLayout. The layout will determine where my textbox sits on the webpage, and I need to agree to an interface, to lock in my textbox positions as soon as possible.
Within group 21, I would have to interact with the GUI interface - Error manager. Since we are strictly not allowed to use exceptions (as it would double our time blindly), I would definitely have to communicate with the Error Manager class to report any and all recoverable errors.

WebButton

Class Description

This class makes a button on a webpage that you can control with c++. Goal is to create the button, change its text, show and hide it, enable or disable, and run c++ code when the button is clicked.

Similar Classes
Std::string - for button labels, css class names

std::function - to store the callback that runs on click

Key functions
Webbutton (const string& label)
Webbutton (const string& label, const string& id)
Std::string Getlabel () const
Void setposition(int x, inty)
Bool isvisible () const
Void setonclick ()
Std:string getID() const

Error Conditions
Programmer error
Assert or clear error message
Duplicate element ids
Invalid label input

Expected Challenges
Avoiding memory leaks
Coordinating layouts ; deciding weather webbuttons controls its own position or weblayout

Related C++ classes of other groups
We will work on WebLayout, WebTextBox and WebCanvas.
In other groups (20) buttons may toggle logging,e xport data, replay modes
Group (21) their button will do the same job just now in a browser
The agent groups their button will trigger simulations

WebCanvas

Class Description

The WebCavas class will be used to manage and operate as a HTML5 canvas object. Canvas objects are a container for graphics. The canvas object usually specifies an ID for what should be drawn in the part of the page it resides in. Javascript is then used to manage the drawing. Instead this class will be handling the drawing as well.

Goals:

- Abstract away the complexity of interfacing between the other C++ classes and the JS canvas API
- Enable efficient drawing/rendering
- Support the different drawing modes for a canvas object like immediate or retained.
- Canvas lifecycle management

- Instantiation
- Sizing
- Clearing
- Transformations

Similar Classes

- std::vector: Managing collections of points, commands, etc.
- std::string: For rendering, file names, and canvas IDs.
- std::optional: Handling optional parameters.
- std::exception: Handling errors nicely.
- std::variant: For managing different drawing commands.
- std::unique_ptr: Might be necessary for holding other classes' objects.
- std::shared_ptr: Might be necessary for holding other classes' objects.
- std::tuple: For passing values that need to go into or be returned from a function together.
- std::function: For the callback mechanism.

Key functions

- WebCanvas(int width, int height, const std::string& id): Creates the canvas
- void Resize(int width, int height): Resizes the canvas to to exact height and width
- void Clear(): Clears what is in the canvas
- void SetBackgroundColor(std::tuple<int, int, int> rgb) or hex code: Sets the background color of the canvas
- void DrawLine(std::pair<double, double> x, std::pair<double, double> point y): Draws a line
- void DrawRect(double x, double y, double width, double height, bool filled): Draws a rectangle
- void DrawCircle(double x, double y, double radius, bool filled): Draws a circle
- void DrawPolygon(const std::vector<std::pair<double, double>>& points, bool filled = true): Draws a polygon
- void DrawText(const std::string& text, double x, double y): Draws some text
- void DrawImage(const std::string& imageId, double x, double y, double width, double height): Draws an image
- void SetPenColor(std::tuple<int, int, int> rgb) or hex code): Sets the pen color
- void SetFillColor(std::tuple<int, int, int> rgb) or hex code): Sets the fill color
- void SetLineWidth(double width): Sets the line width of the pen
- void SetFont(const std::string& font): Sets the font of the canvas
- void SetAlpha(double alpha): Sets the alpha of the canvas
- void Translate(std::pair<double, double> point): Moves the canvas to this point
- void Rotate(double angle, bool clockwise): Rotates the canvas
- void Scale(double x, double y): Scales the canvas
- void Save(): Saves the state of the canvas
- void Restore(): Restores a saved state of a canvas
- int GetWidth() const: returns the width of a canvas

- int GetHeight() const: returns the height of a canvas
- std::string GetCanvasId() const: returns the ID of a canvas
- void RequestAnimationFrame(std::function<void()>callback): Calls the passed function before a redraw.
- std::tuple GetLocation(): Gets the coordinates of the canvas

Error Conditions

Programmer Errors:
- Drawing with negative dimensions.
- Invalid alpha values.
- Restoring without saving.
- Drawing without an initialized canvas.
- Drawing a polygon with less than 3 points.
- Entering RGB values less than 0 and greater than 255.

Recoverable Errors:
- Canvas element creation fails.
- WebGL context creation fails.
- Image loading fails if the resource isn't found.
- Memory allocation failures due to larger operations.
- Emscripten runtime fails when calling functions between the C++ code and Javascript environment.

User Errors
- Drawing outside of the canvas bounds.
- Attempting to use an image that is not loaded yet.
- Using unavailable fonts.
- Using invalid RGB values.

Expected Challenges

- Learning how to use Emscripten to call the Javascript canvas API from C++.
- Coordinating the C++ object lifetimes with the Javascript canvas element lifecycle.
- Reducing overhead and improving performance. I might need to batch animation requests if there are many of them.
- Handling async operations.
- Handling events like mouse clicks and sending them back to C++ callbacks.
- Testing.

Related C++ classes of other groups

Group 21:

Since they are also making a UI, it would be wise to coordinate on common abstractions like color, coordinate systems, and image management. We might be able to use each other's classes for some core functionality.

Agent groups:
Understanding how they represent agent positions and paths will help me design appropriate and efficient rendering methods. I also would like them to easily be able to define what an agent looks like visually and be on their way.

World groups:
Similar to the agent groups. Understanding what they need to draw and how they need it to draw will help me figure out my rendering methods. I would also like them to be able to interface easily.

Group 13:
They have classes like point, box, and circle. So it would be smart to define a common way to represent these objects. I could maybe just use their classes to draw those shapes.

Group 20:
I'd like to work with the data group to create visualizations like graphs, charts, etc.

WebLayout

- Class Description
- Similar Classes
- Key functions
- Error Conditions
- Expected Challenges
- Related C++ classes of other groups

Class Description

WebLayout will be a class that will act as a container to lay out other elements on screen. It will be a simple container encompassing the functionality of a Flex container, allowing children to be arranged along an axis with options for justifying and aligning them. The ability to choose a horizontal or vertical axis and compose multiple WebLayouts together would give us a simple interface while also being flexible enough to create arbitrary 2D layouts that are responsive and allow different ways for child components to space themselves.

Similar Classes

From the standard library, there aren't classes that are directly similar as it does not do GUI or layout. Classes that would be useful to implement this would be

- std::string: Strings could be used to represent many properties for the layout, such as whether the flex direction is “row” or “column”, if align items is “left”, “center”, etc. We may also use enums for this instead to have stronger typing.
- std::vector<T>: This would be helpful as a container for children of the one-directional layout, allowing us to dynamically add and remove elements.
- std::pair and std::tuple: These would be useful for grouping together related values that methods would potentially return.
- std::shared_ptr and std::weak_ptr: These would be useful for holding pointers to child elements that they want to share or point to without claiming ownership of them (depending on how we structure ownership of UI components)

Key Functions

The functions below assume UIElement as a common base class that all UI component classes in the program derive from

- void AddChild(std::weak_ptr<UIElement> elem)
- void RemoveChildAt(size_t index)
- void RemoveChild(std::weak_ptr<UIElement> elem)
- void SetDirection(std::string dir)
- void SetJustifyContent(std::string justify)
- void SetAlignItems(std::string align)
- void SetAlignContent(std::string align)
- void SetWrap(boolean wrap)
- void SetGap(std::string gap)

Error Conditions

User errors would be passing invalid values to the property setters that accept strings, such as SetJustifyContent and SetAlignItems. We could make this a programmer error by switching to enums.
Another user error would be a user adding a child to the WebLayout that it already contains.

Expected Challenges

- How will flex children set properties on themselves that are not a property of the container but the child itself (eg. you can set flex-grow on a child in CSS to decide how much it will take up available space). We could create a WebLayoutChild class such that it would hold the actual child item and would also allow the user to set child flex properties through this class.
- Who owns all the UI components? Should there be a singleton class representing the document such that it manages all the created elements under it, and a WebLayout gets a shared pointer to them? What will be class that represents actually adding this element to the document in the DOM?

Other Groups' CSS classes

We would need to first standardize how we represent elements within our own group to make sure that all classes representing UI elements derive from a common class that WebLayout can contain.
From other groups, I expect to interact with the analytics group to work on the layout of displaying the logs that are captured from the simulation, as they will lead to a layout with a large number of children that we might need to optimize for rendering.

2)

Our Vision

The vision for our main module is to create a simple, accessible UI interface for the other groups that will abstract away aspects of interacting with the DOM. The classes we design will be composable and provide an easy way to construct basic layouts and handle interactions, as well as update them dynamically, without much boilerplate or setup code.

Our main module will be a flexible, web-based user interface system. The module brings together a series of classes to enable a control and visualization system for scientific simulations. We aim to create an intuitive web user interface that makes simulation control, simulation visualization, and data visualization simple and appealing using Emscripten, SDL, and C++.

The web interface will serve as one of the main user-facing components which will provide:

- Dynamic simulation visualization
- Control panel interfaces
- Information displays
- Asset management
- Responsive layout

Overall, responsiveness and ease of use are the two main goals we are reaching for. Following those goals, we seek to achieve good efficiency and an appealing UI.