Group 17: *GUI Interface*

Note: Depending on the GUI library that you choose, some of the options below may already be provided to you. If so, you may either create a better version, substitute them out for one of the alternative classes listed below, or suggestion your own alternative that you believe would be useful for your goals.

Image Manager - Sitara Baxendale

ImageManager - An object that handles image assets by name, pre-loading them (if needed) and displaying them on the screen as requested. This will centralize image handling making it easier for other parts of the program to request an image drawn to the screen

Similar STL Classes:

- std::unordered_map (to map image names to loaded image objects)
- std::string (for image identifiers)

Key Functions:

- load_image(const std::string& name, const std::string& file_path)
- get_image(const std::string& name)
- draw_image(const std::string& name, int x, int y)
- has_image(const std::string& name)

Potential Errors:

Image name not found → **(3) User error** (return false or null)
File cannot be loaded → **(2) Recoverable error** (throw exception)
Duplicate image name → **(1) Programmer error** (assert)

Expected Challenges:

- Figure out how WxWIdgets or SDL deals with images
- Avoiding memory leaks
- Managing ownership and lifetime of image resources

Coordinate with Other Teams:

- Group 14 & 15 (Worlds)

    - They define terrain, tiles, environments → we display them
- Group 1 & 2 (Agents)
    - They define agent visuals (sprites, icons)

In order to have consistent image names / IDs we need to coordinate this

ImageGrid - Deni

ImageGrid - A grid of images (presumably from the ImageManager) that can be displayed as a unit. Simplifies the drawing a World background for the users. Typically agents and items would be layered on top of this ImageGrid.

Similar STL Classes:

- std::vector
- std::array
- std::pair

Key Functions:

- set_cell(int row, int col, const std::string& image_name)
- draw()
- resize(int rows, int cols)
- clear()

Potential Errors:

- Invalid row/column indices → (1) Programmer error (assert)
- Missing image in ImageManager → (3) User error

Expected Challenges:

- Coordinating grid dimensions with screen resolution
- Performance considerations for large grids

Coordinate with Other Teams:

- Image Manager - Grid relies on manager.
- Agent systems are rendered separately and layered on top of ImageGrid

Menu - Anagha Jammalamadaka

Menu - A set of options that can be modified as needed in code. These options should be able to be provided through the GUI to a user, and when one is chosen it should be associated with a function to be called.

Similar STL Classes:

- Std::vector - store menu options in order (start/pause/end)
- Std::function - stores callbacks for button options
- Std::string - store labels for menu buttons ("Start", "Pause", "End", "Next")

Key Functions:

- add_option(const std::string& label, std::function<void()> callback)
    - Adds a new selectable
- remove_option(const std::string& label)
    - Removes an option from name
- draw()
    - Draws the menu buttons, displays the menu buttons
- handle_input(int input_code)
    - Interprets keyboard/mouse input and updates selection or activates an option.

Potential Errors:

- Selecting an invalid option → (3) User error
- Null callback → (1) Programmer error

Expected Challenges:

- Handling user input cleanly
- Mapping input events to menu options
- Layout and spacing of menu items
- Supporting both keyboard and mouse navigation
- Handling highlight state when menu options are being disabled or removed

Coordination:

- Coordinate with Worlds (Groups 14/15) to agree on menu actions like:
    - Load world / reset world / toggle grid / step simulation

- Coordinate with Agents (Groups 1/2) for actions like:
    - spawn agent / pause agents / change agent speed
- Coordinate with ErrorManager so invalid selections can show a warning consistently
- Coordinate on input event definitions (what key means "pause", etc.)

Text - Kiana May

Text - A form of string that also tracks text properties, such as font, color, boldness, size, etc. This class will simplify text rendering in the GUI.

Similar STL Classes:

- std::string

Key Functions:

- set_content(const std::string& text)
- set_color(Color c)
- set_font(const std::string& font_name)
- set_size(int point_size)
- reset_style()
- draw(int x, int y)

Potential Errors:

- Font not found → (2) Recoverable error (throw exception or fall back to default font)
- Negative font sizes → (1) Programmer error (assert)
- Empty text content when drawing → (3) User error

Expected Challenges:

- Text wrapping and alignment if needed
- Supporting Unicode and special characters

Coordination:

- Group 16 Output manager: to display text information to users
- This Group 17 ErrorManager, Menu: will need to use text to display errors

- Group 18 WebTextbox: Should work similarly and share font and color naming to make it simple to switch between text rendering in web and GUI
- Group 1 or 2 Agents: if agents need text bubbles or status

Error Manager - Hugh Mark Sanchez (sanch350)

ErrorManager - A set of tools to manage errors that occur, providing a clear message to the user based on saved functions that the programmer can override. For example, in a terminal, it should print the errors to the screen (perhaps even in color), while in your GUI it might pop up a message on the screen. You could have it more versatile so that if the program is compiled to the web it might either pop up an alert or write to the web console. You could also make the class configurable to specify an alternative action to take when an error occurs (for example, a command-line app might want to intercept the error to use as feedback). You should also be able to have multiple levels of errors (at least separate “warnings” from “fatal errors”, but you may want other levels of severity, debug messages, or something else.)

Similar STL Classes:

- std::cerr / std::clog - Standard error and logging streams
- std::expected<T, E - Contains either a value or an error
- std::optional<T> - Represents optional values
- std::format / std::print - For formatted error messages

Key Functions:

- TerminalError
- GUIError
- LogError
- ClearErrors
- LogWarning

Potential Errors:

- Incompatible Inputs
- File I/O
- Memory Failure - Allocation or access
- Missing Handler - button has no destination

Expected Challenges:

- Gracefully handling errors while avoid using exceptions
- Group coordination for error vs warning
- What platform will we use - widgets vs SDL
    - Different built in options
- Error log format - xml, csv, etc.

Coordination:

- All groups - They all need error reporting, so providing a clean interface is crucial
- Group 1 & 2 (Agents) - May need to report agent-related errors
- Group 14 & 15 (Worlds) - May need to report world loading/validation errors

Vision for the main module (for all classes):

Our GUI Interface module will provide a desktop application for players to interact with the game world. We plan to use SDL (Simple DirectMedia Layer) or WxWidgets to handle rendering and input, with a focus on keeping the interface modular and performant. The main display will show the game world using our ImageGrid class, with agents and items layered on top, alongside UI elements like menus, text displays, and status information.

We'll treat the player as another agent using the Agent API, which keeps our code simpler and avoids duplicating game logic in the GUI. The ImageManager will handle loading and caching image assets so they can be drawn efficiently. Our Menu class will provide flexible options screens that can trigger different game actions. The Text class will handle all text rendering needs, from menu labels to dialogue boxes to status displays, with support for different fonts, colors, and sizes. The ErrorManager will display error messages and warnings to users in a clear way, using the Text class for consistent formatting.

We'll also implement a development mode that allows viewing and editing beyond the normal player perspective. Our five classes are designed to be independent and reusable, making it straightforward for other groups to integrate with our interface. This approach should support different game types and allow for easy iteration as other modules develop their features.