- Module testing

- Purpose: Module testing is a software testing technique where individual components or modules of a program are tested in isolation from the rest of the application. The goal is to verify that each performs as expected.
- The primary goal is to identify and fix defects or issues within individual components before they are integrated into the complete system.
- Managerial independence between the SQA and development team was maintained

- **Import/Export :** The import/Export Panel handles import and export to and for xml File.

1) File Path & Name Validation:
   - isValidPath() detects valid/invalid paths
   - isValidFileName () rejects invalid name
   - is ValidFileName() accepts valid names

Test Passed

2) XML Import
   - Parsing of XML with valid structure
   - Error handling for malformed XML

Test Passed

3) XML Export
   - XML file is created with correct structure
   - State matrix is written correctly

Test Passed

- StateGrid :

Unit testing for this class should verify:

1) Initialization Tests:
   - Everything works properly without error, with a default size of 5.
   - Verify grid creation with custom size of 100
   - State_ matrix is initialized correctly.

   Test Passed

2) Grid Manipulation Tests:
   - pushNewRow() works correctly (forward and reverse)
   - setCellState() updates cells correctly
   - getcellState() return values
   - copystatematrix() creates and accurate copy
   - setStateMatrix update the grid

   Test Passed

3) Edge Case test
   - IndexInRange() handles negative
   - setGridSize() handle invalid
   - setPossibleStates()

   Test Passed

4) Visualization Test:
   - paintComponents() renders correctly in both display formats black_white and numbers
   - paintcell() renders properly
   - initializeRandomGrid() works properly with random initialize with valid states

- resetgrid() clears the grid correctly

Test Passed

5) Mouse Interaction Test:
   - All clicking is working good

Test Passed

- Setting Panel :

1) Initialization Tests:
   - Panel initializes properly with all controls visible: delay slider, grid size spinners, randomize checkbox, and start/reset buttons.
   - Default values are correctly set(e.g., delay = 500 ms, grid size = 5x5)

Test Passed

2) Control Behaviour Tests:
   - Delay slider updates simulation speed correctly.
   - Grid size spinners(widthSpinner, heightSpinner) update the StateGrid dimensions as expected.
   - Randomize checkbox toggles whether the grid starts with random values.

Test Passed

3) Button Functionality Tests:
   - Start button triggers a simulation run in the MainFrame.

- Reset button clears the grid and resets control to defaults.
- All buttons respond on the first click, no lag or UI freeze.

Test Passed

4) Edge Case Tests:

- Grid size spinners reject invalid values(e.g., negative, 0, overly large).
- Delay slider does not allow out of range values.
- Random checkbox retains state across multiple runs.

Test Passed

- StatisticsPanel :

Unit testing for this class should verify:

1)Initialization Tests:

- Panel initializes without errors
- Default labels (“Alive Cells:”, Dead Cells:”) are displayed properly
- Initial counts are both zero before any update

Test Passed

2)Update Functionality Tests:

- updateStatistics() correctly receives a grid matrix and calculate the alive/dead counts.
- Accurate count for different sized grids(5x5, 100x100).
- Tested with grids containing only one alive cell or all cells alive.

Test Passed

3) Edge Case Tests:

- Null or empty matrices handled gracefully without crashing.
- Works correctly with non square grids(e.g., 3x5).

Test Passed

- VisualizationPanel :

Unit testing for this class should verify:

1) Color Selection Tests:

- Color buttons open color picker dialogs correctly
- Alive/dead cell color selections are stored internally.
- getAliveColor() and getDeadColor() return correct selected colors.

Test Passed

2) Grid Color Application Tests:

- Grid reflects new colors immediately after color change.
- Tested with both light and dark color selections for visibility.

Test Passed

3) Edge Case Tests:
   - Handles invalid/null color values (color resets to default).
   - Re-selecting the same color does not break the GUI

Test Passed

- Output Log :

Unit testing for this class should verify:

1) Initialization Tests:
   - The OutputLog panel initializes correctly with a scrollable text area.
   - Initial text area is empty or contains a welcome message.

Test Passed

2) Logging Functionality Tests:
   - log() method appends new messages with proper formatting (e.g., timestamps).
   - Messages persist correctly and appear in the correct order.

- Scroll automatically follows new entries if enabled.

Test Passed

3) Integration Tests:
   - Successfully receives logs from import/export events, rule application, and errors.
   - Log updates immediately upon triggered events in other components.

Test Passed

4) Ege Case Tests:
   - Handles long log messages and special characters.
   - Log remains stable when flooded with high-frequency messages.
   - No duplication or data loss when multiple messages are logged in succession.

Test Passed

- TabParentclass :

Unit testing for this class should verify:

1) InitializationTests:
   - Initializes with correct tab layout.
   - All added components (StateGrid, OutputLog, etc.) appear under the correct tabs.

Test Passed

2) Tab Management Tests:
   - Tabs can be dynamically added or removed if applicable.
   - Correct tab is selected and shown when switched by the user.

Test Passed

3) UI Behaviour Tests:
   - Switching between tabs maintains the state of each component (e.g., grid remains unchanged when switching tabs).
   - Tabs display correct icons or names if customized.

Test Passed

4) Edge Case Test:
   - Handles null or empty tab components without crashing.
   - Layout remains stable during dynamic resizing or addition/removal of tabs.
   - Prevents duplicate tab entries.

Test Passed

- StatusBar :

1) Initialize Test
   - StatusBar initialized with default “ Ready Status”
   - Verified UI component (label alignment, border)

Test Passed

2) Status Update Test
    - setStatus() updates the label text

Test Passed

3) UI Rendering Test
    - Verify Label is left aligned
    - Verify border is visible

Test Passed

- Integration testing

Test #1 UI <-> StateGrid:

Test Description: Test for if user clicks will reflect changes on the StateGrid
Testing Steps: Click on any cell while simulation is playing
Test Desired Output: The cell will update its state/colour
Test Results: Cell updated state/colour correctly upon user click
Test Passed

Test #2 UI <-> StateGrid:

Test Description: Test the Switch tab feature functionality
Testing Steps: Switch from Settings tab to Visualization tab, and then to the Statistics tab
Test Desired Output: The GUI will display the Settings tab, then the Visualization tab, and finally the Statistics tab
Test Results: The GUI correctly displayed the Settings/Visualization/Statistics tab based on which one was clicked
Test Passed

Test #3 SimulationSettings <-> RuleExecutionBackend:

Test Description: Test the simulations Pause/Play functionality
Testing Steps: Populate random cells in the grid and then Play/Pause the simulation
Test Desired Output: The simulation should be running when the state is set to “Active”, and if the simulation is paused when the state is set to “Paused”

Test Results: The Simulation properly plays or paused when the state is set to “Active” or “Paused”
Test Passed

Test #4 SimulationSettings <-> RuleExecutionBackend:

Test Description: Test for the set simulation speed functionality
Testing Steps: While the grid is populated and the simulation is running, set different speeds using the provided slider tool
Test Desired Output: The simulation speed should grow as the slider is increased and it should slow down as the slider is decreased
Test Results: The Simulation properly increased/decreased in speed depending on the position of the slider tool
Test Passed

Test #5 RuleExecutionBackend <-> CARule:

Test Description: Test for the 1D/2D simulation rule functionality
Testing Steps: Set the simulation rule to 1D/2D, and then run the simulation
Test Desired Output: Simulation should correctly run, following the corresponding 1D/2D simulation format structure
Test Results: The simulation correctly outputs the cells according to the selected 1D/2D rule
Test Passed

Test #6 CARule <-> StateGrid:

Test Description: Test for correct computing of the next state of the current grid
Testing Steps: Populate cells within the grid, click on Next button to compute the next state of the current grid
Test Desired Output: Simulation should correctly output the next state according to the cells currently active on the grid
Test Results: The simulation correctly computes the next state of the current grid
Test Passed

Test #7 Import/Export <-> Settings + StateGrid:

Test Description: Test for importing grid states into the current workspace
Testing Steps: Select a valid file to import under the Import/Export tab and import it
Test Desired Output: The settings should be correctly aligned with that of the imported file, and the stategrid should output the correct state
Test Results: The settings and stategrid were both successfully implemented based off of the imported value

Test Passed

Test #8 Import/Export <-> Settings + StateGrid:

Test Description: Test for exporting grid states from the current workspace
Testing Steps: Populate the stategrid, set setting conditions and export the current workspace as a valid XML file
Test Desired Output: The workspace should be saved as an XML file in the directory of the users choice, containing all the features of the workspace
Test Results: The settings and stategrid were both successfully exported as an XML file onto the users local device, ready to be imported in other uses of the application
Test Passed

Test #9 Statistics <-> StateGrid:

Test Description: Test for correct functionality of our Dead/Alive/% of grid used/Population statistics
Testing Steps: Populate the state grid, then run it and check the statistics tab
Test Desired Output: The Dead/Alive/% of grid used/Population lines under the statistics tab should be correctly updated in real time
Test Results: The Dead/Alive/% of grid used/Population lines under the statistics tab were correctly implemented and successfully update in real time, based off of the stategrid
Test Passed

Test #10 VisualizationSettings <-> UI:

Test Description: Test for cell color change scheme functionality
Testing Steps: Populate the state grid, then set preferred colour settings under the visualization tab and run the simulation
Test Desired Output: The colour of the cells displayed to the user should correctly match that of the inputs they set in the visualization tab
Test Results: The cells were correctly matched to their corresponding colour settings that were set in the visualization tab
Test Passed

- Test case testing

Purpose: Test case testing checks that the software does what it should by running it through a set of clear examples. Glass-box testing is a type of test case testing where we design tests by looking directly at the code's inner logic

Rule Settings: User is able to select 1D or 2D grid

Testing Passed

1D Rule Number: Only allows the user to enter an integer between 0-255

Testing Passed

Grid Size: User is only able to select a grid size between 5-100 via slider

Testing Passed

Simulation Speed: User is only able to select a simulation speed between 1-15 via slider

Testing Passed

Cell Display Format: Allows the user to select black and white or number format

Testing Passed

Active/Dead Cell Color: User input is able to change the color of live and dead cells

Testing Passed