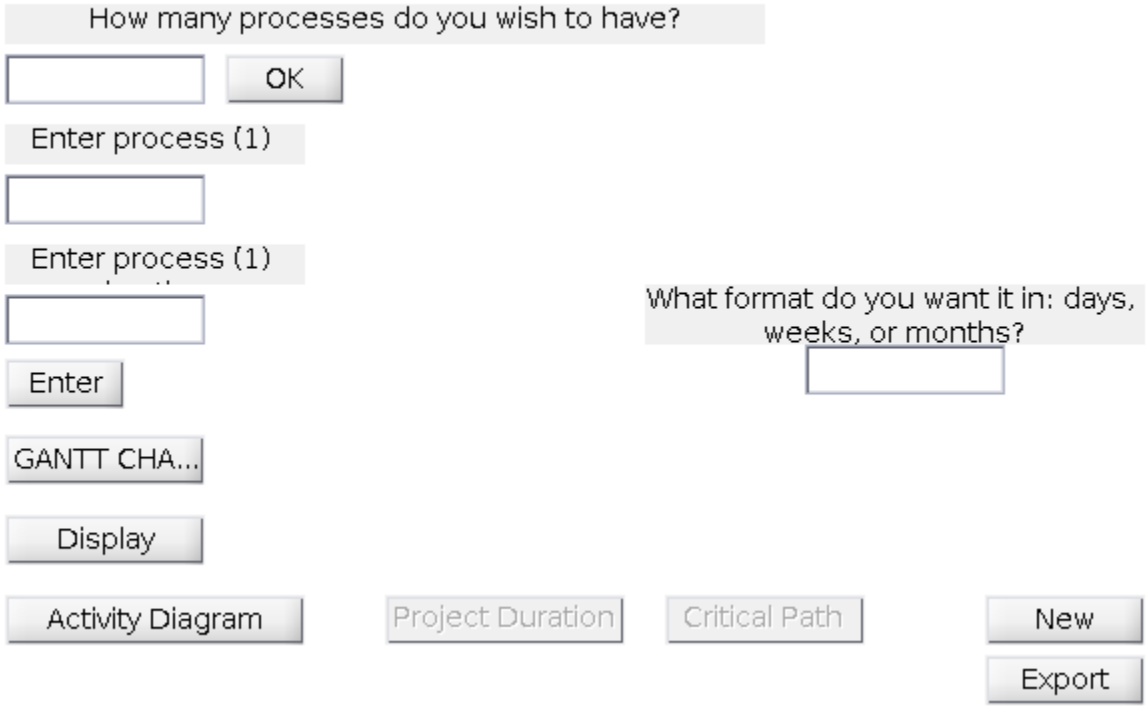
```
%publish('FullWorkinOne', 'format', 'pdf', 'outputDir', '/MATLAB Drive/');

function processSchedulerGUI()

 % this is the main GUI
mainFig = figure('Name', 'Process Scheduler', 'Position', [100, 100, 600,
400], 'NumberTitle', 'off', 'MenuBar', 'none', 'ToolBar', 'none');
mainFig.Color = [0.1, 0.35, 0.7];
num_task = 0;
processname = cell(1, num_task);
processduration = zeros(1, num_task);
predecessors = cell(1, num_task);


%the first menu: Number of processes
uicontrol('Style', 'text', 'String', 'How many processes do you wish to
have?', 'Position', [10, 340, 380, 20]);
numTaskTextBox = uicontrol('Style', 'edit', 'String', '', 'Position', [10,
310, 100, 25]);
uicontrol('Style', 'pushbutton', 'String', 'OK', 'Position', [120, 310, 60,
25], 'Callback', @setNumTasks);

%the second menu: Enter process details
processIndex = 1;
count = 1;

uicontrol('Style', 'text', 'String', ['Enter process (', num2str(count), ')
name:'], 'Position', [10, 280, 150, 20], 'UserData', 'processNameLabel');
processNameTextBox = uicontrol('Style', 'edit', 'String', '', 'Position',
[10, 250, 100, 25]);

uicontrol('Style', 'text', 'String', ['Enter process (', num2str(count),
') duration:'], 'Position', [10, 220, 150, 20], 'UserData',
'processDurationLabel');
processDurationTextBox = uicontrol('Style', 'edit', 'String', '', 'Position',
[10, 190, 100, 25]);


enterButton = uicontrol('Style', 'pushbutton', 'String', 'Enter', 'Position',
[10, 158, 60, 25], 'Callback', @enterProcess);

calculationButton = uicontrol('Style', 'pushbutton', 'String', 'GANTT CHART',
'Position', [10, 120, 100, 25], 'Callback', @calculateGantt);
displayButton = uicontrol('Style', 'pushbutton', 'String', 'Display',
'Position', [10, 80, 100, 25], 'Callback', @display);
activityDiagramButton = uicontrol('Style', 'pushbutton', 'String', 'Activity
Diagram', 'Position', [10, 40, 150, 25], 'Callback', @drawActivityDiagram);

projectDurationButton = uicontrol('Style', 'pushbutton', 'String',
'Project Duration', 'Position', [200, 40, 120, 25], 'Callback',
@calculateProjectDuration, 'Enable', 'off');
criticalPathButton = uicontrol('Style', 'pushbutton', 'String', 'Critical
Path', 'Position', [340, 40, 100, 25], 'Callback', @calculateCriticalPath,
```

```
'Enable', 'off');
newButton = uicontrol('Style', 'pushbutton', 'String', 'New', 'Position',
[500, 40, 80, 25], 'Callback', @newButtonClick);
uicontrol('Style', 'text', 'String', 'What format do you want it in: days,
weeks, or months?', 'Position', [330, 190, 250, 30]);
formatTextBox = uicontrol('Style', 'edit', 'String', '', 'Position', [410,
165, 100, 25]);


tableData = magic(3);
columnNames = {'Column1', 'Column2', 'Column3'};


exportButton = uicontrol('Style', 'pushbutton', 'String', 'Export',
'Position', [500, 10, 80, 25], 'Callback', @exportData);

setappdata(gcf, 'tableData', tableData);
setappdata(gcf, 'columnNames', columnNames);
```




```
%AUBREY created this funct?on
    function setNumTasks(~, ~)
        num_task = str2double(get(numTaskTextBox, 'String'));
        if isnan(num_task) || mod(num_task, 1) ~= 0 || num_task <= 0
            errordlg('Please enter a valid positive integer for the number of
processes.', 'Error', 'modal');
            return;
        end

        set(numTaskTextBox, 'Enable', 'off');
        set(processNameTextBox, 'Enable', 'on');
        set(processDurationTextBox, 'Enable', 'on');
        set(enterButton, 'Enable', 'on');
    end

%this function checks to make sure a process is valid
    function isValid = isValidProcess(processName)
        isValid = any(strcmp(processName, processname));

        if processIndex > 1
            if processIndex <= numel(predecessors) &&
~isempty(predecessors{processIndex - 1})
                predNames = strsplit(get(predecessors{processIndex - 1},
'String'), ',');

                invalidPredecessors = setdiff(predNames, processname);

                if ~isempty(invalidPredecessors)
```

```
                    isValid = false;
                    errordlg(['Invalid predecessor(s): ',
strjoin(invalidPredecessors, ', '), '. Enter valid processes first.'],
'Error', 'modal');
                end
            end
        end

% this part prevents the process name be entered to not be the same
        if processIndex <= numel(processname) && strcmp(processName,
processname{processIndex})
            isValid = false;
            errordlg('A process cannot be its own predecessor. Enter a valid
predecessor.', 'Error', 'modal');
        end

        if ~isValid
            errordlg(['Invalid process "', processName, '". Enter a valid
process first.'], 'Error', 'modal');
        end
    end

%function to enter a new process and save it
  function enterProcess(~, ~)
        processName = get(processNameTextBox, 'String');
        processDuration = str2double(get(processDurationTextBox, 'String'));

        if isempty(processName) || isnan(processDuration) || processDuration
<= 0
            errordlg('Please enter valid process name and duration.',
'Error', 'modal');
            return;
        end

        if any(strcmp(processName, processname))
            errordlg(['Process "', processName, '" is already entered.'],
'Error', 'modal');
            return;
        end

         processname{processIndex} = processName;
        processduration(processIndex) = processDuration;

        processIndex = processIndex + 1;

         set(processNameTextBox, 'String', '');
        set(processDurationTextBox, 'String', '');

        % Enable predecessor entry after the first process name is entered
        if processIndex > 1
           if isValidProcess(processName)
                addPredecessorInput();
            end
        end
```

```
         if processIndex > num_task
            set(enterButton, 'Enable', 'off');
            set(projectDurationButton, 'Enable', 'on');
            set(criticalPathButton, 'Enable', 'on');
        end

        %to update count for process name
        count = count + 1;
        set(findobj('Style', 'text', 'String', ['Enter process (',
num2str(count - 1), ') name:']), 'String', ['Enter process (',
num2str(count), ') name:']);

         if processIndex > 1
            updatePredecessorLabel();
        end

        % Update count for process duration
        updateDurationLabel();
    end

  function addPredecessorInput()

        uicontrol('Style', 'text', 'String', ['Enter process (',
num2str(count), ') predecessor(s):'], 'Position', [180, 250, 180, 35],
'UserData', 'predecessorsLabel');
        predecessorsTextBox = uicontrol('Style', 'edit', 'String', '',
'Position', [180, 200, 100, 25], 'UserData', 'predecessorsTextBox');

         predecessors{processIndex - 1} = predecessorsTextBox;

         updatePredecessorLabel();
    end

%function to update predecessor label
    function updatePredecessorLabel()
        %locates the current value of the predecessor label and set it to
        %an incremental value (+1) when ENTER button is pressed
        set(findobj('Style', 'text', 'String', ['Enter process (',
num2str(count - 1), ') predecessor(s):']), 'String', ['Enter process (',
num2str(count), ') predecessor(s):']);
    end

   function updateDurationLabel()
        %locates the current value of the process duration label and set
        % it to an incremental value (+1) when ENTER button is pressed
        set(findobj('Style', 'text', 'String', ['Enter process (',
num2str(count - 1), ') duration:']), 'String', ['Enter process (',
num2str(count), ') duration:']);
    end

%function to draw Gant Chart
    function calculateGantt(~, ~)
        %this checks if all processes have been entered
        if isempty(processname) || numel(processname) ~= num_task
```

```
            %numel = number of elements
            %if the number of elements are not the same as the number
            %entered in the numTaskTextBox, show this error
            errordlg('Please enter all processes before calculating the Gantt
chart.', 'Error', 'modal');
            return;
        end

        selectedFormat = get(formatTextBox, 'String');
        %means it should be taken to see if a string has been entered
        if isempty(selectedFormat) %selected format should not be empty
            errordlg('Please enter the desired format (days, weeks, or
months).', 'Error', 'modal');
            return;
        end
   startTimes = zeros(1, num_task);
        for i = 1:num_task
            %if number of tasks is not more than 1, it is not a gant chart
            if i > 1
               %strsplit seperates each string entered by a comma
                % if user seperates 2 process names by a comma, system
                % automatically understands it as 2 predecessors
                predecessorNames = strsplit(get(predecessors{i-1}, 'String'),
',');

                predecessorIndices = find(ismember(processname,
predecessorNames));

                if ~isempty(predecessorIndices)
                    startTimes(i) = max(startTimes(predecessorIndices) +
processduration(predecessorIndices));
                end
            end
        end


        ganttFig = figure('Name', 'Gantt Chart', 'Position', [100, 100, 800,
400], 'NumberTitle', 'off');
                                %ganttFig.Color= [0.9,0.7,0.2];

        endTimes = startTimes + processduration;

        %generate Gantt chart
        hold on; %this prevents a process bar from replacing previous bar
        for i = 1:num_task

             durationText = sprintf(' %g %s', processduration(i),
selectedFormat);
            rectangle('Position', [startTimes(i), i-0.4, processduration(i),
0.8], 'FaceColor', [0.5 0.5 0.5]);

            %to display the duration next to the bar
            text(endTimes(i), i, durationText, 'VerticalAlignment', 'middle');
        end
```

```
      set(gca, 'YTick', 1:num_task, 'YTickLabel', processname);
        xlabel('Time');
        ylabel('Processes');
        title('Gantt Chart');

        grid on;

        hold off;
    end

%this function draw acctivity diagram using the process and duration
%entered
    function drawActivityDiagram(~, ~)
        if isempty(processname) || numel(processname) ~= num_task
            errordlg('Please enter all processes before drawing the activity
diagram.', 'Error', 'modal');
            return;
        end

        startTimes = calculateStartTimes();

        activityFig = figure('Name', 'Activity Diagram', 'Position', [300,
300, 800, 400], 'NumberTitle', 'off');
        hold on;

        grid on;
        set(gca, 'GridColor', 'k', 'GridAlpha', 0.5);

        radius = 0.2;
        for i = 1:num_task
            x = startTimes(i) + processduration(i) / 2;
            y = i;

            theta = linspace(0, 2*pi, 100);
            xPoints = x + radius * cos(theta);
            yPoints = y + radius * sin(theta);

             plot(xPoints, yPoints, 'k-', 'LineWidth', 2);
            text(x, y + 0.5, processname{i}, 'HorizontalAlignment', 'center');

           if ~isempty(predecessors{i}.String)
                dependentTasks = strsplit(predecessors{i}.String, ',');
                for j = 1:length(dependentTasks)
                    dependentIndex = find(strcmp(processname,
dependentTasks{j}));
                    if ~isempty(dependentIndex)
                        xDependent = startTimes(dependentIndex) +
processduration(dependentIndex) / 2;
                        yDependent = dependentIndex;

                         arrowStart = [x + radius, y];
                        arrowEnd = [xDependent - radius, yDependent];
```

```
                        annotation('arrow', 'X', [arrowStart(1),
arrowEnd(1)], 'Y', [arrowStart(2), arrowEnd(2)], 'LineWidth', 2, 'Color',
'g');
                    end
                end
            end
        end

         legend(processname, 'Location', 'Best');


        xlabel('Time');
        ylabel('Processes');
        title('Activity Diagram');
        hold off;
    end

function startTimes = calculateStartTimes()

        startTimes = zeros(1, num_task);
        for i = 1:num_task
            if i > 1
                 predecessorNames = strsplit(get(predecessors{i-1},
'String'), ',');
                predecessorIndices = find(ismember(processname,
predecessorNames));

                 if ~isempty(predecessorIndices)
                    startTimes(i) = max(startTimes(predecessorIndices) +
processduration(predecessorIndices));
                end
            end
        end
    end

   function calculateProjectDuration(~, ~)

       ganttFig = findobj('Name', 'Gantt Chart');

         selectedFormat = get(formatTextBox, 'String');
        if isempty(selectedFormat)
            errordlg('Please enter the desired format (days, weeks, or
months).', 'Error', 'modal');
            return;
        end

        if isempty(ganttFig)
            errordlg('Please calculate the Gantt chart before determining the
Project Duration.', 'Error', 'modal');
            return;
        end

        textObjects = findobj(ganttFig, 'Type', 'text');
```

```
       if isempty(textObjects)
            errordlg('No processes found in the Gantt chart.', 'Error',
'modal');
            return;
        end

         maxEndTime = 0;

        for i = 1:numel(textObjects)
           position = textObjects(i).Position;

             maxEndTime = max(maxEndTime, position(1));
        end

        msgbox(['Project Duration: ', num2str(maxEndTime), ' ',
selectedFormat], 'Project Duration', 'modal');
    end

  function calculateCriticalPath(~, ~)
        if isempty(processname) || numel(processname) ~= num_task
            errordlg('Please enter all processes before calculating the
Critical Path.', 'Error', 'modal');
            return;
        end

       predecessorsCopy = predecessors;

         dependencyMatrix = zeros(num_task);
        for i = 1:num_task
            if ~isempty(predecessorsCopy{i}.String)
                dependentTasks = strsplit(predecessorsCopy{i}.String, ',');
                for j = 1:length(dependentTasks)
                   dependentTaskNum = find(strcmp(processname,
dependentTasks{j}));
                    if isempty(dependentTaskNum)
                        errordlg(['Invalid predecessor "', dependentTasks{j},
'" for task ', num2str(i), '.'], 'Error', 'modal');
                        return;
                    end
                    dependencyMatrix(dependentTaskNum, i) = 1;
                end
            end
        end

         startingTasks = find(sum(dependencyMatrix, 1) == 0);

        [earliestStartTime, earliestFinishTime] =
calculateEarlyTimes(startingTasks, dependencyMatrix);

        criticalPathIndices = find(earliestFinishTime ==
max(earliestFinishTime));

         criticalPathNames = processname(criticalPathIndices);
        criticalPathString = ['CRITICAL PATH IS: ',
```

```
strjoin(criticalPathNames, ' -> ')];
        msgbox(criticalPathString, 'Critical Path', 'modal');
    end

  function [earliestStartTime, earliestFinishTime] =
calculateEarlyTimes(startingTasks, dependencyMatrix)
        numTasks = numel(startingTasks);
        earliestStartTime = zeros(1, numTasks);
        earliestFinishTime = zeros(1, numTasks);

        for i = 1:numTasks
            [earliestStartTime, earliestFinishTime] =
calculateTimes(startingTasks(i), earliestStartTime, earliestFinishTime,
dependencyMatrix);
        end
    end

   function [earliestStartTime, earliestFinishTime] = calculateTimes(node,
earliestStartTime, earliestFinishTime, dependencyMatrix)
        neighbors = find(dependencyMatrix(:, node));
        for i = 1:length(neighbors)
            if earliestFinishTime(neighbors(i)) < earliestStartTime(node) +
processduration(node)
                earliestFinishTime(neighbors(i)) = earliestStartTime(node) +
processduration(node);
            end
            [earliestStartTime, earliestFinishTime] =
calculateTimes(neighbors(i), earliestStartTime, earliestFinishTime,
dependencyMatrix);
        end
    end

 function order = toposort(adjMatrix)
        numNodes = size(adjMatrix, 1);
        visited = false(1, numNodes);
        order = zeros(1, numNodes);
        index = numNodes;

        for i = 1:numNodes
            if ~visited(i)
                [order, index] = dfs(i, adjMatrix, visited, order, index);
            end
        end
    end

   function [order, index] = dfs(node, adjMatrix, visited, order, index)
        visited(node) = true;
        neighbors = find(adjMatrix(:, node));
        for i = 1:length(neighbors)
            if ~visited(neighbors(i))
                [order, index] = dfs(neighbors(i), adjMatrix, visited, order,
index);
            end
        end
        order(index) = node;
```

```
        index = index - 1;
    end

  function display(~, ~)
        if isempty(processname) || numel(processname) ~= num_task
            errordlg('Please enter all processes before displaying the
information.', 'Error', 'modal');
            return;
        end

        infoFig = figure('Name', 'Process Information', 'NumberTitle', 'off');

        screenSize = get(0, 'ScreenSize');
        figWidth = 600; figHeight = 150 + 20 * num_task + 20;
        figPosition = [(screenSize(3) - figWidth) / 2, (screenSize(4) -
figHeight) / 2, figWidth, figHeight];
        set(infoFig, 'Position', figPosition);
        infoFig.Color = [0.2, 0.5, 0.1];

        predecessorInfo = cell(1, num_task);
        for i = 1:num_task
            if i == 1
                predInfo = ['No predecessor for ', processname{i}];
            else
                if ~isempty(predecessors{i-1}.String)
                    predecessorNames = strsplit(predecessors{i-1}.String,
',');
                    [~, predIndices] = ismember(predecessorNames,
processname);

                    [~, sortIndices] = sort(predIndices);
                    sortedPredecessorNames = predecessorNames(sortIndices);

                    %to display predecessor information
                    if isempty(sortedPredecessorNames)
                        predInfo = ['No predecessor for ', processname{i}];
                    else
                        predInfo = ['Predecessor for ', processname{i}, ': ',
strjoin(sortedPredecessorNames, ', ')];
                    end
                else
                    predInfo = ['No predecessor for ', processname{i}];
                end
            end

            predecessorInfo{i} = predInfo;
        end

        tableData = [processname', num2cell(processduration'),
predecessorInfo'];

        table = uitable('Data', tableData, 'ColumnName', {'PROCESS NAME',
'DURATION', 'PREDECESSORS'}, 'Position', [10, 40, 580, 20 * num_task + 20]);
```

```
        columnWidth = {'auto', 'auto', 300};
        set(table, 'ColumnWidth', columnWidth);
    end

 function exportData(~, ~)
       fig = gcf;
        tableData = getappdata(fig, 'tableData');
        columnNames = getappdata(fig, 'columnNames');

        if isempty(tableData) || isempty(columnNames)
            errordlg('No data available for export.', 'Export Error');
            return;
        end

          [fileName, pathName] = uiputfile('*.xlsx', 'Save As');

       if isequal(fileName, 0) || isequal(pathName, 0)
            return;
        end

         filePath = fullfile(pathName, fileName);

        try
             xlswrite(filePath, [columnNames; num2cell(tableData)]);
            msgbox('Data exported successfully.', 'Export Success');
        catch
            errordlg('Error exporting data to Excel.', 'Export Error');
        end
    end

%this function is used to export data to PDF
    function exportToPDF(pdfFileName, dataTable, columnNames)
        pdfDoc = mlreportgen.pdf.PDFDocument(pdfFileName);

         addTitle(pdfDoc, 'Process Data Export');

         addTable(pdfDoc, dataTable, 'Header', columnNames, 'Style', 'grid');

         close(pdfDoc);
    end

 %restarting the textboxes and labels
    function newButtonClick(~, ~)
      set(numTaskTextBox, 'String', '');
        set(processNameTextBox, 'String', '');
        set(processDurationTextBox, 'String', '');
        set(formatTextBox, 'String', '');

          clearGanttDisplay();

         num_task = 0;
        processname = cell(1, num_task);
        processduration = zeros(1, num_task);
        predecessors = cell(1, num_task);
```

```
         set(numTaskTextBox, 'Enable', 'on');
        set(processNameTextBox, 'Enable', 'off');
        set(processDurationTextBox, 'Enable', 'off');
        set(enterButton, 'Enable', 'off');
        set(projectDurationButton, 'Enable', 'off');
        set(criticalPathButton, 'Enable', 'off');

        processIndex = 1;
        count = 1;

         processNameLabel = findobj(mainFig, 'Style', 'text', 'UserData',
'processNameLabel');
        processDurationLabel = findobj(mainFig, 'Style', 'text', 'UserData',
'processDurationLabel');

        set(processNameLabel, 'String', ['Enter process (', num2str(count),
') name:']);
        set(processDurationLabel, 'String', ['Enter process (',
num2str(count), ') duration:']);

        predecessorsLabels = findobj(mainFig, 'Style', 'text', 'UserData',
'predecessorsLabel');
        set(predecessorsLabels, 'Visible', 'off');

         predecessorsTextBoxes = findobj(mainFig, 'Style', 'edit',
'UserData', 'predecessorsTextBox');
        set(predecessorsTextBoxes, 'Visible', 'off');
    end

   function clearGanttDisplay()
              ganttFig = findobj('Name', 'Gantt Chart');
        if ~isempty(ganttFig)
            close(ganttFig);
        end

        displayFig = findobj('Name', 'Process Information');
        if ~isempty(displayFig)
            close(displayFig);
        end

        set(projectDurationButton, 'Enable', 'off');
        set(criticalPathButton, 'Enable', 'off');
    end

end
```

Published with MATLAB® R2024a