Name: Julie Ann Gunayon Delda
Student Number: 82334569
Course: Model-based Conceptual Design and Innovation

Prompt: *Give 3 Assumptions in developing a block-based programming learning tool for high school students.*

Assumption 1: **Familiarity with Basic Technology**: Students are assumed to have basic familiarity with computers, such as navigating an interface, using a mouse and keyboard, and understanding fundamental computer operations (e.g., saving files, dragging and dropping). This allows the tool to focus on teaching programming concepts rather than basic computer literacy.

Assumption 2: **Gradual Learning Curve**: It's assumed that students will benefit from a gradual learning curve, where they start with foundational concepts (like sequences, loops, and conditionals) and progressively advance to more complex topics. The tool is designed to accommodate this by introducing simple concepts first and building on them, reducing cognitive load.

Assumption 3: **Visual Learning Preference**: High school students are assumed to benefit from a visual learning approach that simplifies code syntax and reduces errors, making programming more accessible and engaging. Block-based tools capitalize on this by using colorful blocks and drag-and-drop mechanics, which help students focus on logic without the burden of syntax errors.

Concept Model

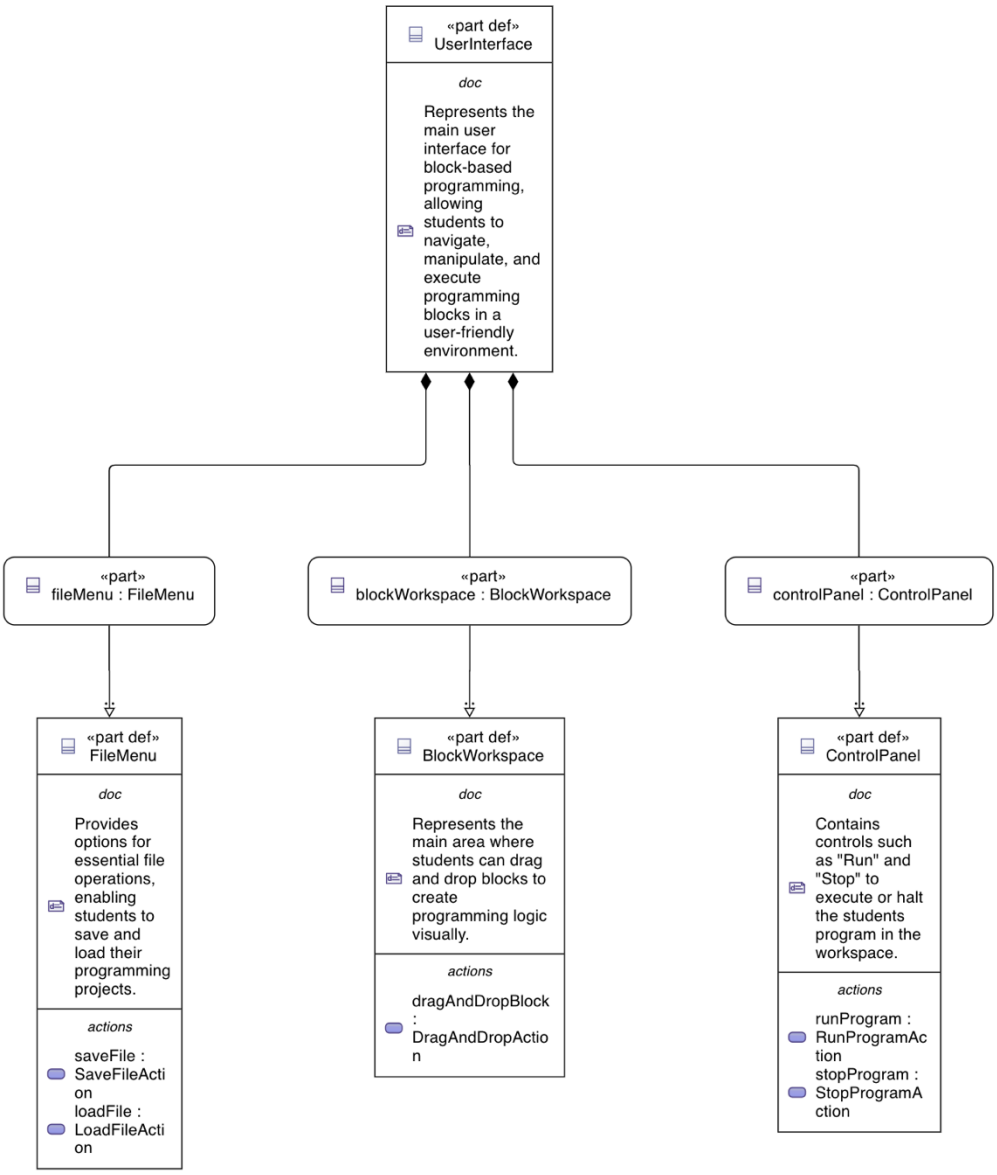
Prompt: *Given the following statement for a product named "block-based programming learning tool": Students are assumed to have basic familiarity with computers, such as navigating an interface, using a mouse and keyboard, and understanding fundamental computer operations (e.g., saving files, dragging and dropping). This allows the tool to focus on teaching programming concepts rather than basic computer literacy. Can you propose a series of parts and functions to cover the realization of this statement?*

```
package 'BlockProgrammingLearningTool' {

    package 'LearningToolComponents' {

        // Parts for user interface components and interactions
        part def UserInterface {
            doc /* Represents the main user interface for block-based programming, allowing
students to navigate,
                    manipulate, and execute programming blocks in a user-friendly environment. */

            part fileMenu: FileMenu;
            part blockWorkspace: BlockWorkspace;
```

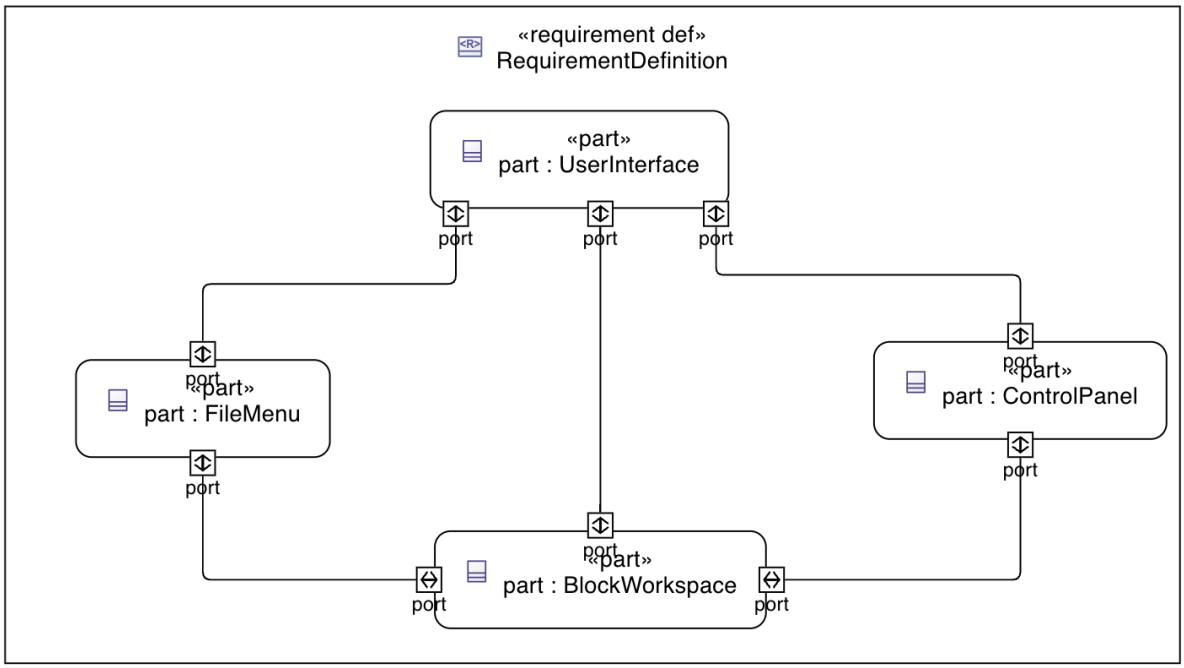
```
        part controlPanel: ControlPanel;
    }

    part def FileMenu {
        doc /* Provides options for essential file operations, enabling students to save and load their programming projects. */
        action saveFile: SaveFileAction;
        action loadFile: LoadFileAction;
    }

    part def BlockWorkspace {
        doc /* Represents the main area where students can drag and drop blocks to create programming logic visually. */
        action dragAndDropBlock: DragAndDropAction;
    }

    part def ControlPanel {
        doc /* Contains controls such as "Run" and "Stop" to execute or halt the student's program in the workspace. */
        action runProgram: RunProgramAction;
        action stopProgram: StopProgramAction;
    }

    // Action Definitions
    action def SaveFileAction {
        doc /* Defines the save operation, allowing students to store their current workspace configuration to a file. */
    }

    action def LoadFileAction {
        doc /* Defines the load operation, enabling retrieval of previously saved workspace configurations. */
    }

    action def DragAndDropAction {
        doc /* Supports drag-and-drop functionality, allowing blocks to be repositioned within the workspace to build logic. */
    }

    action def RunProgramAction {
        doc /* Initiates the execution of the program created in the block workspace. */
    }

    action def StopProgramAction {
```

```
        doc /* Halts the execution of the program, returning the workspace to a non-running state. */
      }
   }

   package 'ToolBehaviors' {
      import LearningToolComponents::*;

      // Behavioral sequence covering file handling and program execution
      state 'ToolUsageStates' parallel {

         state 'FileOperationStates' {
            doc /* Represents the sequence of states associated with file operations, including saving and loading files. */

            state ready;
            state saving;
            state loading;

            transition 'ready-to-saving'
               first ready
               accept 'saveFile'
               do SaveFileAction
               then saving;

            transition 'saving-to-ready'
               first saving
               then ready;

            transition 'ready-to-loading'
               first ready
               accept 'loadFile'
               do LoadFileAction
               then loading;

            transition 'loading-to-ready'
               first loading
               then ready;
         }

         state 'ProgrammingExecutionStates' {
            doc /* Represents the sequence of states for controlling program execution, from idle to running to stopped. */
```

```
            state idle;
            state running;
            state stopped;

            transition 'idle-to-running'
               first idle
               accept 'runProgram'
               do RunProgramAction
               then running;

            transition 'running-to-stopped'
               first running
               accept 'stopProgram'
               do StopProgramAction
               then stopped;

            transition 'stopped-to-idle'
               first stopped
               then idle;
         }
      }
   }
}
```



Concept Model for Block-based Programming Tool



Interconnection View

Proof of Concept Model

Prompt: *Can you create a Proof of Concept Model for me based on the latest concept model? Pleas continue to add documentation and do not use "perform" but rather use "action".*

Prompt: *Can you suggest parts on the market that I could use to realize the model? Can you incorpate it in the Proof of Concept Model?*

```
package 'BlockProgrammingLearningTool_PoC' {

    package 'LearningToolComponents_PoC' {

        // Primary interface and interaction components
        part def UserInterface {
            doc /* Proof of Concept for the user interface of the Block-Based Programming Learning
Tool,
                    covering file operations, workspace manipulation, program control, and feedback. */

            part fileMenu: FileMenu {
                doc /* Part of the UI for handling save and load actions. */
            }
            part blockWorkspace: BlockWorkspace {
```

```
            doc /* Part of the UI for the workspace where students arrange programming blocks.
*/
        }
        part controlPanel: ControlPanel {
            doc /* Part of the UI for running and stopping the program. */
        }
        part helpPanel: HelpPanel {
            doc /* Provides guidance, tutorials, or tooltips for using the tool effectively. */
        }
        part settingsPanel: SettingsPanel {
            doc /* Allows customization of tool settings and preferences. */
        }
        part feedbackPanel: FeedbackPanel {
            doc /* Displays program feedback and error messages for debugging. */
        }
        part toolbox: Toolbox {
            doc /* Contains programming blocks that students can drag into the workspace. */
        }
        part console: Console {
            doc /* Outputs program execution results and debug information. */
        }
    }

    // Physical hardware components used to realize the tool
    part def ProcessingUnit {
        doc /* Central processing unit to handle logic and software execution for the tool. */
        part raspberryPi4: RaspberryPi4 {
            doc /* Raspberry Pi 4 Model B, affordable and capable for educational coding
applications. */
        }
        part intelNUC: IntelNUC {
            doc /* Intel NUC for enhanced processing power, ideal for larger setups or more
complex applications. */
        }
    }

    part def Display {
        doc /* Touchscreen display allowing user interaction with the block programming
interface. */
        part waveshareTouchDisplay: WaveshareTouchDisplay {
            doc /* Waveshare 7-inch capacitive touchscreen, suitable for individual or small-group
coding exercises. */
        }
        part dellTouchMonitor: DellTouchMonitor {
```

```
            doc /* Dell 24" P2418HT touch monitor, providing a larger workspace for interactive learning. */
        }
    }

    part def SensorsAndInputs {
        doc /* Interactive input components that enhance engagement with the programming tool. */
        part circuitPlaygroundExpress: CircuitPlaygroundExpress {
            doc /* Adafruit Circuit Playground Express, with built-in sensors and inputs for interactive programming. */
        }
        part leapMotionController: LeapMotionController {
            doc /* Leap Motion Controller for gesture-based interaction, adding a hands-free input method. */
        }
    }

    part def Storage {
        doc /* Storage options for saving and loading programming projects. */
        part usbDrive: SanDiskDualDrive {
            doc /* SanDisk Ultra Dual USB drive for cross-device compatibility and project portability. */
        }
        part externalSSD: WDMyPassportSSD {
            doc /* WD My Passport SSD for fast storage access, especially for larger projects. */
        }
    }

    part def InterfaceDevices {
        doc /* Additional devices that allow for effective interaction and control within the tool. */
        part keyboardTouchpad: LogitechK600 {
            doc /* Logitech K600 TV Keyboard with integrated touchpad, suitable for navigating the interface. */
        }
        part streamDeck: ElgatoStreamDeck {
            doc /* Elgato Stream Deck with customizable buttons, providing quick access to common tool actions. */
        }
    }

    part def AudioVisualFeedback {
```

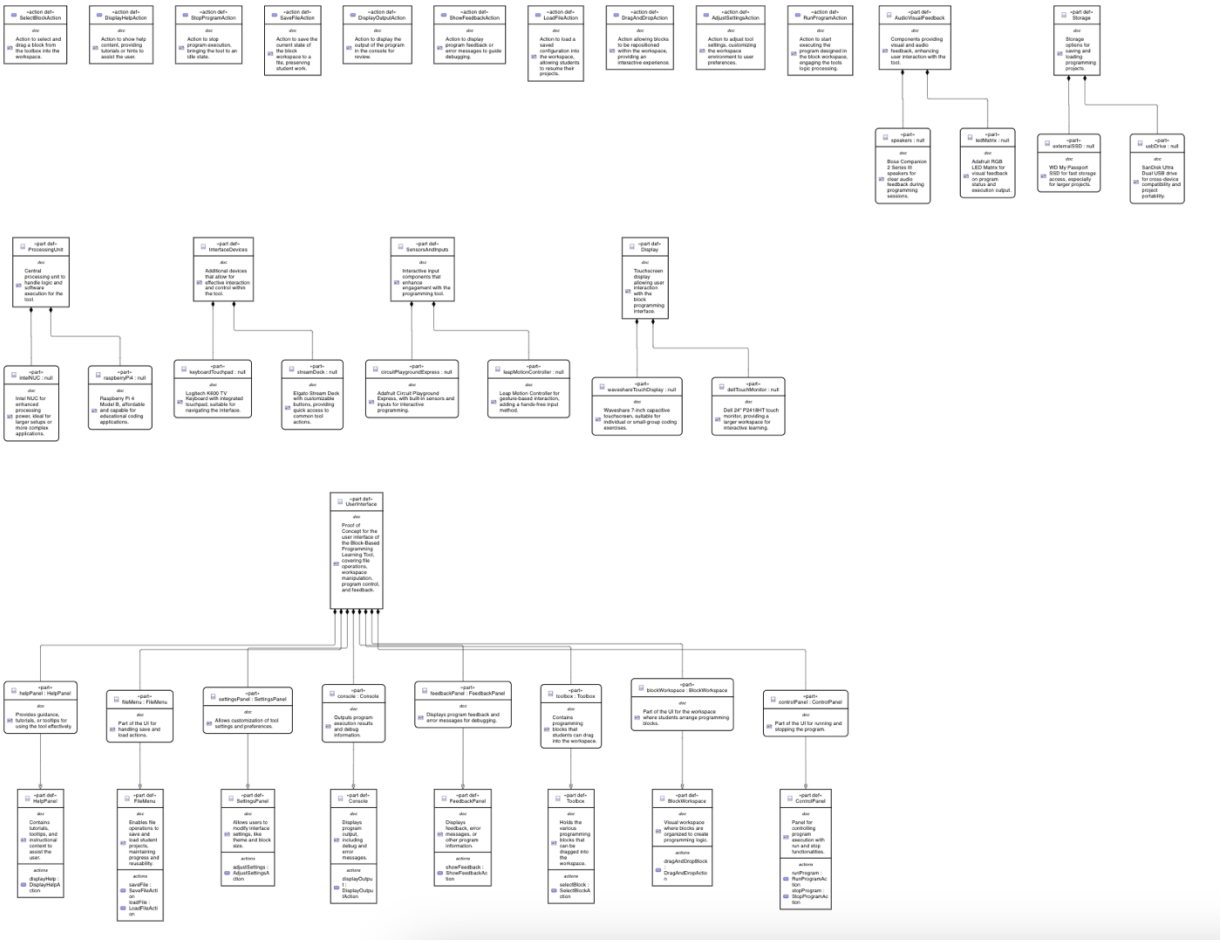
```
        doc /* Components providing visual and audio feedback, enhancing user interaction with the tool. */
        part speakers: BoseCompanionSpeakers {
            doc /* Bose Companion 2 Series III speakers for clear audio feedback during programming sessions. */
        }
        part ledMatrix: AdafruitLEDMatrix {
            doc /* Adafruit RGB LED Matrix for visual feedback on program status and execution output. */
        }
    }

    // Defining each of the User Interface parts and their actions
    part def FileMenu {
        doc /* Enables file operations to save and load student projects, maintaining progress and reusability. */
        action saveFile: SaveFileAction;
        action loadFile: LoadFileAction;
    }

    part def BlockWorkspace {
        doc /* Visual workspace where blocks are organized to create programming logic. */
        action dragAndDropBlock: DragAndDropAction;
    }

    part def ControlPanel {
        doc /* Panel for controlling program execution with run and stop functionalities. */
        action runProgram: RunProgramAction;
        action stopProgram: StopProgramAction;
    }

    part def HelpPanel {
        doc /* Contains tutorials, tooltips, and instructional content to assist the user. */
        action displayHelp: DisplayHelpAction;
    }

    part def SettingsPanel {
        doc /* Allows users to modify interface settings, like theme and block size. */
        action adjustSettings: AdjustSettingsAction;
    }

    part def FeedbackPanel {
        doc /* Displays feedback, error messages, or other program information. */
        action showFeedback: ShowFeedbackAction;
```

```
    }

    part def Toolbox {
        doc /* Holds the various programming blocks that can be dragged into the workspace. */
        action selectBlock: SelectBlockAction;
    }

    part def Console {
        doc /* Displays program output, including debug and error messages. */
        action displayOutput: DisplayOutputAction;
    }

    // Action Definitions with Documentation
    action def SaveFileAction {
        doc /* Action to save the current state of the block workspace to a file, preserving
student work. */
    }

    action def LoadFileAction {
        doc /* Action to load a saved configuration into the workspace, allowing students to
resume their projects. */
    }

    action def DragAndDropAction {
        doc /* Action allowing blocks to be repositioned within the workspace, providing an
interactive experience. */
    }

    action def RunProgramAction {
        doc /* Action to start executing the program designed in the block workspace, engaging
the tool's logic processing. */
    }

    action def StopProgramAction {
        doc /* Action to stop program execution, bringing the tool to an idle state. */
    }

    action def DisplayHelpAction {
        doc /* Action to show help content, providing tutorials or hints to assist the user. */
    }

    action def AdjustSettingsAction {
        doc /* Action to adjust tool settings, customizing the workspace environment to user
preferences. */
```

```
    }

    action def ShowFeedbackAction {
      doc /* Action to display program feedback or error messages to guide debugging. */
    }

    action def SelectBlockAction {
      doc /* Action to select and drag a block from the toolbox into the workspace. */
    }

    action def DisplayOutputAction {
      doc /* Action to display the output of the program in the console for review. */
    }
  }

  package 'ToolBehaviors_PoC' {
    import LearningToolComponents_PoC::*;

    // State-based behavior for the tool, addressing file and execution workflows
    state 'ToolUsageStates' parallel {
      doc /* Proof of Concept behavior states demonstrating basic interaction workflows:
           file operations, program execution control, and feedback display. */

      state 'FileOperationStates' {
        doc /* Covers file actions such as saving and loading, representing student interaction
with project files. */

        state ready {
          doc /* State when the tool is ready to perform file operations. */
        }
        state saving {
          doc /* Temporary state while saving a project. */
        }
        state loading {
          doc /* Temporary state while loading a project into the workspace. */
        }

        transition 'ready-to-saving'
          first ready
          accept 'saveFile'
          do SaveFileAction
          then saving {
          doc /* Transition triggered by 'saveFile' action, leading to the saving state. */
        }
```

```
    transition 'saving-to-ready'
        first saving
        then ready {
        doc /* Transition from saving back to ready state after save completes. */
    }

    transition 'ready-to-loading'
        first ready
        accept 'loadFile'
        do LoadFileAction
        then loading {
        doc /* Transition triggered by 'loadFile' action, leading to the loading state. */
    }

    transition 'loading-to-ready'
        first loading
        then ready {
        doc /* Transition from loading back to ready state after load completes. */
    }
}

state 'ProgrammingExecutionStates' {
    doc /* Manages program execution states such as idle, running, and stopped. */

    state idle {
        doc /* State when the tool is inactive, waiting to run a program. */
    }
    state running {
        doc /* Active state where the program is currently running. */
    }
    state stopped {
        doc /* State after stopping the program, ready to restart or edit. */
    }

    transition 'idle-to-running'
        first idle
        accept 'runProgram'
        do RunProgramAction
        then running {
        doc /* Transition to running state on 'runProgram' action. */
    }

    transition 'running-to-stopped'
```

```
            first running
            accept 'stopProgram'
            do StopProgramAction
            then stopped {
            doc /* Transition to stopped state when 'stopProgram' action is accepted. */
        }

        transition 'stopped-to-idle'
            first stopped
            then idle {
            doc /* Transition from stopped back to idle, allowing further interaction. */
        }
    }
  }
}
```



Proof of Concept Model 1

Prompt: *Create a Traceability Table showing elements modified from the concept model to the proof of concept model.*

Below is a Traceability Table that captures the modifications made to the "Block-Based Programming Learning Tool" as the model evolved from the initial concept to the proof of concept. This table highlights new elements, updates, and additional hardware components incorporated into the proof of concept model.

Element	Type	Modification	Description
`UserInterface`	Part	Added detailed subcomponents for feedback and settings	Expanded UI structure in the proof of concept, adding `HelpPanel`, `SettingsPanel`, `FeedbackPanel`, `Toolbox`,

			and `Console` for a richer interface.
`FileMenu`	Part	Retained from concept model	Handles file operations like saving and loading, unchanged.
`blockWorkspace`	Part	Retained from concept model	Main area for block programming; retained without changes.
`ControlPanel`	Part	Retained from concept model	For program run/stop actions; retained with no modifications.
`HelpPanel`	Part	Added to proof of concept	New part providing tutorials and tooltips for user guidance in the proof of concept model.
`SettingsPanel`	Part	Added to proof of concept	New part enabling customization of settings, like workspace layout and preferences.
`FeedbackPanel`	Part	Added to proof of concept	New part to show feedback, error messages, or program information for debugging purposes.
`Toolbox`	Part	Added to proof of concept	New part holding draggable programming blocks for building logic.
`Console`	Part	Added to proof of concept	New part for displaying output, debug information, and execution results.
`ProcessingUnit`	Part	Added hardware-specific options	New part added in proof of concept with subcomponents `raspberryPi4` and `intelNUC` to provide processing capabilities.
`Display`	Part	Added display hardware options	New part added to specify display components (`waveshareTouchDisplay` and `dellTouchMonitor`) for interactive programming.
`SensorsAndInputs`	Part	Added to support additional inputs	Added in proof of concept with parts `circuitPlaygroundExpress` and `leapMotionController`

			for enhanced user interaction.
`Storage`	Part	Added storage hardware options	New part in proof of concept including `SanDiskDualDrive` and `WDMyPassportSSD` to handle project saving and loading.
`InterfaceDevices`	Part	Added input devices	New part added with `LogitechK600` and `ElgatoStreamDeck` for ease of navigation and programmable shortcuts.
`AudioVisualFeedback`	Part	Added AV feedback devices	New part with `BoseCompanionSpeakers` and `AdafruitLEDMatrix` for visual and auditory feedback during program execution.
`SaveFileAction`	Action	Retained from concept model	Action to save projects, retained with no changes.
`LoadFileAction`	Action	Retained from concept model	Action to load projects, retained without modifications.
`DragAndDropAction`	Action	Retained from concept model	Retained action for block manipulation in the workspace.
`RunProgramAction`	Action	Retained from concept model	Unchanged action to start program execution.
`StopProgramAction`	Action	Retained from concept model	Unchanged action to stop program execution.
`DisplayHelpAction`	Action	Added in proof of concept	New action to show help content, enhancing usability and support for beginners.
`AdjustSettingsAction`	Action	Added in proof of concept	New action for adjusting tool settings, such as layout and interface options.
`ShowFeedbackAction`	Action	Added in proof of concept	New action to display feedback or error messages, assisting in debugging.
`SelectBlockAction`	Action	Added in proof of concept	New action for selecting and dragging blocks from the toolbox into the workspace.
`DisplayOutputAction`	Action	Added in proof of concept	New action to display program output in the

			console, aiding in testing and debugging.

Prompt: *Can you update this as a proof of concept where available parts in the market is incorporated?*

```
package 'BlockProgrammingLearningTool' {

    package 'LearningToolComponents' {

        // Parts for user interface components and interactions
        part def UserInterface {
            doc /* Represents the main user interface for block-based programming, allowing students to navigate,
                    manipulate, and execute programming blocks in a user-friendly environment. */

            part fileMenu: FileMenu {
                part saveButton: SaveButton_LG;
                part loadButton: LoadButton_LG;
            }
            part blockWorkspace: BlockWorkspace {
                part toolbox: BlocklyToolbox;
                part editorArea: MonacoEditor;
            }
            part controlPanel: ControlPanel {
                part runButton: RunButton_Adafruit;
                part stopButton: StopButton_Adafruit;
            }
        }

        // FileMenu components based on commercially available UI elements
        part def FileMenu {
            doc /* Provides options for essential file operations, enabling students to save and load their programming projects. */
            action saveFile: SaveFileAction;
            action loadFile: LoadFileAction;

            part saveButton: SaveButton_LG {
                doc /* Save button using LG's button component for electronics projects. */
                attribute model: String = "LG SK1-24";
                attribute buttonType: String = "IconButton";
            }
```

```
        part loadButton: LoadButton_LG {
            doc /* Load button from LG, compatible with various educational toolkits. */
            attribute model: String = "LG LK9-13";
            attribute buttonType: String = "IconButton";
        }
    }

    // BlockWorkspace with widely used, market-available components
    part def BlockWorkspace {
        doc /* Represents the main area where students can drag and drop blocks to create
programming logic visually. */
        action dragAndDropBlock: DragAndDropAction;

        part toolbox: BlocklyToolbox {
            doc /* Toolbox using Blockly, a popular library for block-based programming. */
            attribute toolType: String = "Blockly";
            attribute version: String = "1.20210923"; // Sample version of Blockly
        }
        part editorArea: MonacoEditor {
            doc /* Code editor area using Monaco Editor, popular in IDEs and online code editors.
*/
            attribute editorType: String = "TextEditor";
            attribute model: String = "Monaco 0.32.0"; // Version commonly used in development
tools
        }
    }

    // ControlPanel with executable control components available in the market
    part def ControlPanel {
        doc /* Contains controls such as "Run" and "Stop" to execute or halt the student's
program in the workspace. */
        action runProgram: RunProgramAction;
        action stopProgram: StopProgramAction;

        part runButton: RunButton_Adafruit {
            doc /* Run button from Adafruit, often used in maker projects. */
            attribute model: String = "Adafruit Mini Push Button";
            attribute buttonType: String = "PhysicalButton";
        }
        part stopButton: StopButton_Adafruit {
            doc /* Stop button, compatible with most educational hardware setups. */
            attribute model: String = "Adafruit Large Push Button";
            attribute buttonType: String = "PhysicalButton";
        }
```

```
    }

    // Action Definitions
    action def SaveFileAction {
        doc /* Defines the save operation, allowing students to store their current workspace
configuration to a file. */
    }

    action def LoadFileAction {
        doc /* Defines the load operation, enabling retrieval of previously saved workspace
configurations. */
    }

    action def DragAndDropAction {
        doc /* Supports drag-and-drop functionality, allowing blocks to be repositioned within
the workspace to build logic. */
    }

    action def RunProgramAction {
        doc /* Initiates the execution of the program created in the block workspace. */
    }

    action def StopProgramAction {
        doc /* Halts the execution of the program, returning the workspace to a non-running
state. */
    }
  }

  package 'ToolBehaviors' {
    import LearningToolComponents::*;

    // Behavioral sequence covering file handling and program execution
    state 'ToolUsageStates' parallel {

      state 'FileOperationStates' {
        doc /* Represents the sequence of states associated with file operations, including
saving and loading files. */

          state ready;
          state saving;
          state loading;

          transition 'ready-to-saving'
            first ready
```

```
            accept 'saveFile'
            do SaveFileAction
            then saving;

        transition 'saving-to-ready'
            first saving
            then ready;

        transition 'ready-to-loading'
            first ready
            accept 'loadFile'
            do LoadFileAction
            then loading;

        transition 'loading-to-ready'
            first loading
            then ready;
    }

    state 'ProgrammingExecutionStates' {
        doc /* Represents the sequence of states for controlling program execution, from idle
to running to stopped. */

        state idle;
        state running;
        state stopped;

        transition 'idle-to-running'
            first idle
            accept 'runProgram'
            do RunProgramAction
            then running;

        transition 'running-to-stopped'
            first running
            accept 'stopProgram'
            do StopProgramAction
            then stopped;

        transition 'stopped-to-idle'
            first stopped
            then idle;
    }
}
```

```
    }
}
```

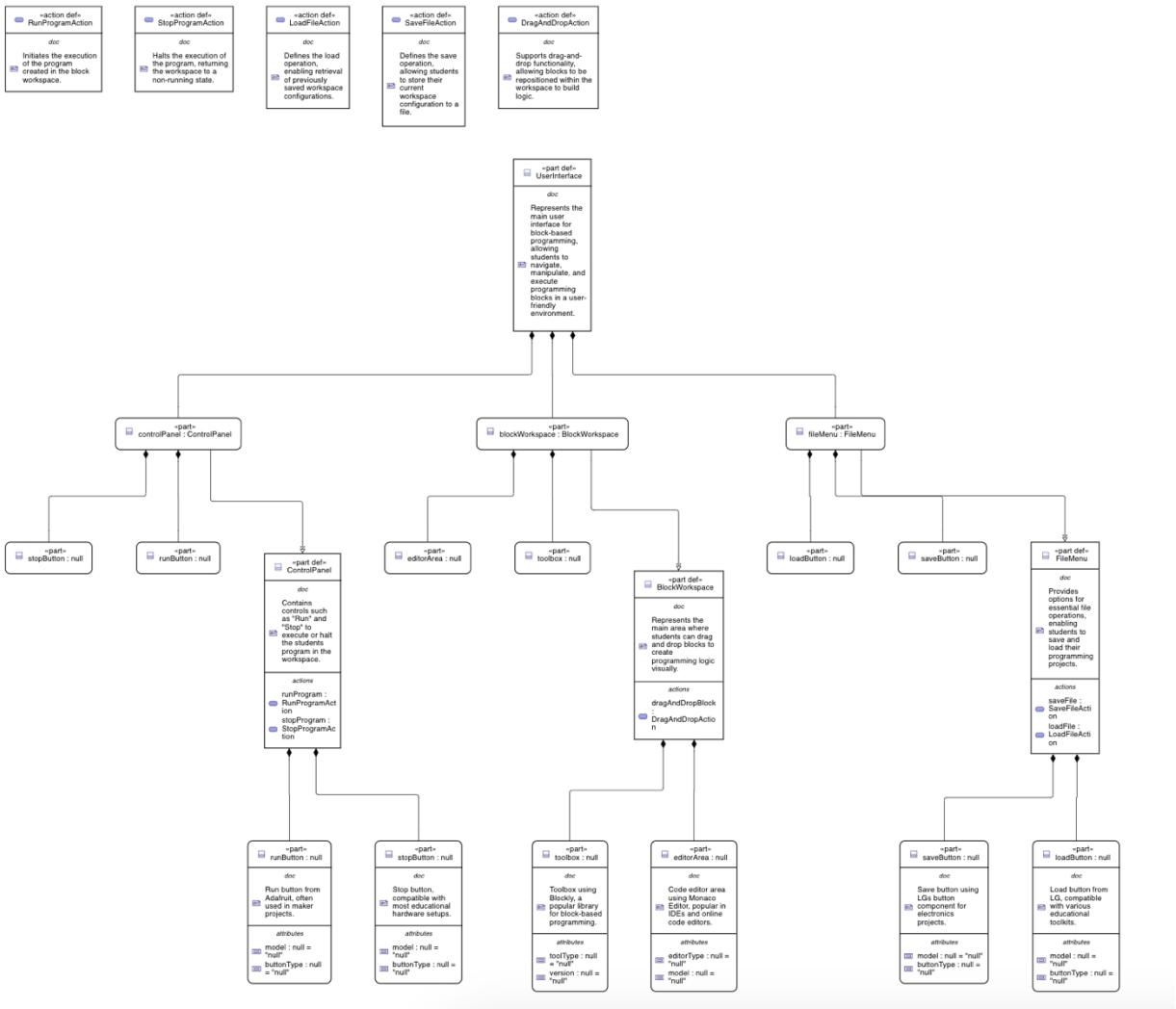
Prompt: *Can you update this as a proof of concept where available parts in the market is incorporated?*

```
package 'BlockProgrammingLearningTool' {

    package 'LearningToolComponents' {

        // Parts for user interface components and interactions
        part def UserInterface {
            doc /* Represents the main user interface for block-based programming, allowing students to navigate,
                    manipulate, and execute programming blocks in a user-friendly environment. */

            part fileMenu: FileMenu {
                part saveButton: SaveButton_LG;
                part loadButton: LoadButton_LG;
            }
            part blockWorkspace: BlockWorkspace {
                part toolbox: BlocklyToolbox;
                part editorArea: MonacoEditor;
            }
            part controlPanel: ControlPanel {
                part runButton: RunButton_Adafruit;
                part stopButton: StopButton_Adafruit;
            }
        }

        // FileMenu components based on commercially available UI elements
        part def FileMenu {
            doc /* Provides options for essential file operations, enabling students to save and load their programming projects. */
            action saveFile: SaveFileAction;
            action loadFile: LoadFileAction;

            part saveButton: SaveButton_LG {
                doc /* Save button using LG's button component for electronics projects. */
                attribute model: String = "LG SK1-24";
                attribute buttonType: String = "IconButton";
            }
            part loadButton: LoadButton_LG {
                doc /* Load button from LG, compatible with various educational toolkits. */
                attribute model: String = "LG LK9-13";
                attribute buttonType: String = "IconButton";
            }
```

```
    }

    // BlockWorkspace with widely used, market-available components
    part def BlockWorkspace {
        doc /* Represents the main area where students can drag and drop blocks to create
programming logic visually. */
        action dragAndDropBlock: DragAndDropAction;

        part toolbox: BlocklyToolbox {
            doc /* Toolbox using Blockly, a popular library for block-based programming. */
            attribute toolType: String = "Blockly";
            attribute version: String = "1.20210923"; // Sample version of Blockly
        }
        part editorArea: MonacoEditor {
            doc /* Code editor area using Monaco Editor, popular in IDEs and online code editors.
*/
            attribute editorType: String = "TextEditor";
            attribute model: String = "Monaco 0.32.0"; // Version commonly used in development
tools
        }
    }

    // ControlPanel with executable control components available in the market
    part def ControlPanel {
        doc /* Contains controls such as "Run" and "Stop" to execute or halt the student's
program in the workspace. */
        action runProgram: RunProgramAction;
        action stopProgram: StopProgramAction;

        part runButton: RunButton_Adafruit {
            doc /* Run button from Adafruit, often used in maker projects. */
            attribute model: String = "Adafruit Mini Push Button";
            attribute buttonType: String = "PhysicalButton";
        }
        part stopButton: StopButton_Adafruit {
            doc /* Stop button, compatible with most educational hardware setups. */
            attribute model: String = "Adafruit Large Push Button";
            attribute buttonType: String = "PhysicalButton";
        }
    }

    // Action Definitions
    action def SaveFileAction {
```

```
        doc /* Defines the save operation, allowing students to store their current workspace configuration to a file. */
    }

    action def LoadFileAction {
        doc /* Defines the load operation, enabling retrieval of previously saved workspace configurations. */
    }

    action def DragAndDropAction {
        doc /* Supports drag-and-drop functionality, allowing blocks to be repositioned within the workspace to build logic. */
    }

    action def RunProgramAction {
        doc /* Initiates the execution of the program created in the block workspace. */
    }

    action def StopProgramAction {
        doc /* Halts the execution of the program, returning the workspace to a non-running state. */
    }
  }

  package 'ToolBehaviors' {
    import LearningToolComponents::*;

    // Behavioral sequence covering file handling and program execution
    state 'ToolUsageStates' parallel {

      state 'FileOperationStates' {
        doc /* Represents the sequence of states associated with file operations, including saving and loading files. */

          state ready;
          state saving;
          state loading;

          transition 'ready-to-saving'
            first ready
            accept 'saveFile'
            do SaveFileAction
            then saving;
```

```
            transition 'saving-to-ready'
               first saving
               then ready;

            transition 'ready-to-loading'
               first ready
               accept 'loadFile'
               do LoadFileAction
               then loading;

            transition 'loading-to-ready'
               first loading
               then ready;
         }

         state 'ProgrammingExecutionStates' {
            doc /* Represents the sequence of states for controlling program execution, from idle
to running to stopped. */

            state idle;
            state running;
            state stopped;

            transition 'idle-to-running'
               first idle
               accept 'runProgram'
               do RunProgramAction
               then running;

            transition 'running-to-stopped'
               first running
               accept 'stopProgram'
               do StopProgramAction
               then stopped;

            transition 'stopped-to-idle'
               first stopped
               then idle;
         }
      }
   }
}
```



Proof of Concept Model 2

Prompt: *Can you create a table showing how the elements in concept model was updated in elements in proof of concept*

Concept Model Element	Proof of Concept Element	Market-Available Component Details
UserInterface	**UserInterface**	Main interface for user interaction
`fileMenu`	`fileMenu`	Provides options for file operations
`FileMenu.saveButton`	`FileMenu.saveButton: SaveButton_LG`	LG SK1-24, IconButton
`FileMenu.loadButton`	`FileMenu.loadButton: LoadButton_LG`	LG LK9-13, IconButton
BlockWorkspace	**BlockWorkspace**	Main area for block-based programming
`BlockWorkspace.toolbox`	`BlockWorkspace.toolbox: BlocklyToolbox`	Blockly, version 1.20210923
`BlockWorkspace.editorArea`	`BlockWorkspace.editorArea: MonacoEditor`	Monaco Editor, version 0.32.0
ControlPanel	**ControlPanel**	Contains controls for program execution
`ControlPanel.runButton`	`ControlPanel.runButton: RunButton_Adafruit`	Adafruit Mini Push Button
`ControlPanel.stopButton`	`ControlPanel.stopButton: StopButton_Adafruit`	Adafruit Large Push Button
Actions	**Actions**	Functional elements remained as is, no change
`SaveFileAction`	`SaveFileAction`	Defines save operation functionality
`LoadFileAction`	`LoadFileAction`	Defines load operation functionality
`DragAndDropAction`	`DragAndDropAction`	Enables drag-and-drop functionality
`RunProgramAction`	`RunProgramAction`	Initiates program execution
`StopProgramAction`	`StopProgramAction`	Halts program execution
ToolBehaviors	**ToolBehaviors**	Behavioral states remain the same
`FileOperationStates`	`FileOperationStates`	Manages save/load file operations
`ProgrammingExecutionStates`	`ProgrammingExecutionStates`	Manages program execution states

List of Prompts

Prompt: *Give 3 Assumptions in developing a block-based programming learning tool for high school students.*

Prompt: *Given the following statement for a product named "block-based programming learning tool": Students are assumed to have basic familiarity with computers, such as navigating an interface, using a mouse and keyboard, and understanding fundamental computer operations (e.g., saving files, dragging and dropping). This allows the tool to focus on teaching programming concepts rather than basic computer literacy. Can you propose a series of parts and functions to cover the realization of this statement?*

Prompt: *Can you create a Proof of Concept Model for me based on the latest concept model? Pleas continue to add documentation and do not use "perform" but rather use "action".*

Prompt: *Can you suggest parts on the market that I could use to realize the model? Can you incorpate it in the Proof of Concept Model?*

Prompt: *Create a Traceability Table showing elements modified from the concept model to the proof of concept model.*

Prompt: *Can you update this as a proof of concept where available parts in the market is incorporated?*

Prompt: *Can you create a table showing how the elements in concept model was updated in elements in proof of concept*

Prompt: *Add available parts I can use in the market*

Prompt: *Can you update this as a proof of concept where available parts in the market is incorporated?*