**Phase 1: Update icons.js**

You mentioned the new "actions" feature uses more icons. The icons.js file you provided already contains SVG definitions for a full set of dice (icon-die-d4, d6, d8, d10, d12, d20, d100) and a bonus icon (icon-die-bonus).

My plan assumes this file is already complete. If the new "actions" file requires *even more* icons that are not in the file I have, this phase would involve adding their <symbol> definitions inside the <svg> tag.

**Phase 2: Update npc engineer.html**

This phase will involve two main changes: adding the new "Actions" card structure and replacing the contents of the "Hit Dice" modal.

1. **Integrate "Actions" Card:** I will take the HTML structure for the new "Actions" card (from the file you provide) and insert it into the main content column, placing it between the "Traits" card and the "Description" card.
2. **Update "Hit Dice" Modal:** I will modify the hp-modal div.
   * I'll remove the old divs containing the hp-num-dice (Number of Dice) input, hp-die-type (Die Type) select, and hp-bonus (Bonus) input.
   * In their place, I will insert the new reusable "dice icon input" UI. This will likely be a new set of divs containing:
     + A display area to show the current dice string (e.g., "3d8 + 6").
     + A series of buttons, each using an <svg> tag to render the icons for d4, d6, d8, d10, d12, d20, and bonus.

**Phase 3: Update style.css**

This file will be updated to ensure the new "actions" feature and the updated modal are styled correctly.

1. **Add "Actions" CSS:** I will add any new CSS classes required by the "Actions" card and its child elements (e.g., styles for an individual action item, its edit/delete buttons, etc.), based on the new file.
2. **Style Dice Input UI:** I will add new CSS rules to style the dice icons in the HP modal. This will make them look like clickable buttons (e.g., cursor: pointer), add hover and active states, and ensure they are arranged neatly with the dice string display.

**Phase 4: Update main.js**

This file will get the core logic for the "actions" feature and the reusable dice input component.

1. **Update defaultNPC:** I will add a new actions: [] property to the baseDefaultNPC object. This will be the array where all of an NPC's actions are stored.
2. **Update healBestiary:** I will add if (!Array.isArray(npc.actions)) npc.actions = []; to the healing logic to ensure all existing NPCs and bestiaries are compatible with the new "actions" data structure.
3. **Integrate "Actions" Logic:** I will add the main functions from your new file. This will include:
   * Functions for adding, editing, and deleting actions from the activeNPC.actions array.
   * The core reusable function(s) for the dice input UI. I plan to export these on the window.app object so ui.js can call them (e.g., window.app.createDiceInput(...)).
4. **Update updateActiveNPCFromForm:** I will add logic to this function to read the data from the new "Actions" card UI and save it to the activeNPC.actions array, just as it does for traits.
5. **Update processTraitString:** The function for replacing tokens like {he} and {name} is very useful. The "Actions" descriptions will almost certainly need this. I'll modify the "Actions" logic to run its description text through this processTraitString function.

**Phase 5: Update ui.js**

This file will be updated to hook up all the event listeners and data-binding for the new UI elements.

1. **Add "Actions" UI Elements:** I will add all the new element IDs (e.g., add-action-btn, action-list-div) from the "Actions" card to the window.ui element map.
2. **Add "Actions" Event Listeners:** In setupEventListeners, I'll add all the listeners for the "Actions" card (e.g., click on "Add Action"). These listeners will call the new functions in main.js.
3. **Update updateFormFromActiveNPC:** I'll add a new section here. This logic will clear the "Actions" UI and then loop through the activeNPC.actions array, creating and appending the HTML for each action to display it on the form.
4. **Update "Hit Dice" Modal Logic:**
   * **setupEventListeners:** I will add click listeners for the new dice icons in the hp-modal. These will call the reusable dice input logic from main.js.
   * **hpApplyBtn Listener:** I will rewrite this listener. It will no longer read from the old inputs. Instead, it will call a function (from main.js) to get the final dice string (e.g., "3d8+6") from the new dice UI and calculate the average HP. It will then apply this to the npc-hp input.
   * **parseHpStringToModal:** I will rewrite this function completely. It will now parse the HP string and then call a "set" function on the new dice UI to make it display the correct number of dice and bonus when the modal opens.

**Phase 6: Update viewport.js**

Finally, this will update the statblock preview to show the actions.

1. **Render "Actions":** In updateViewport, I will add a new section just after the traitsHtml is generated.
2. This new code will loop through the activeNPC.actions array. For each action, it will generate the D&D-style statblock HTML (e.g., <div class="npctop" style="..."><p><i><b>Action Name.</b></i> Action description...</p></div>).
3. This block of "actions" HTML will be inserted into the final statblock template, right after the traits and before the final divider.

**Is there anywhere else the dice input could be used?**

You are right, the **Hit Dice modal** is the most logical place to reuse this component.

The "Actions" feature itself will (presumably) already be using it for damage rolls. The only other places that might accept dice rolls are free-text fields like **Traits** and the main **Description**.

However, this new component sounds like a *structured input* (for building a single dice string), not a text-insertion tool for a rich-text editor. Therefore, the HP modal is the *only* other place I see where this component fits.