Layout Cache Problems in Visual Editor

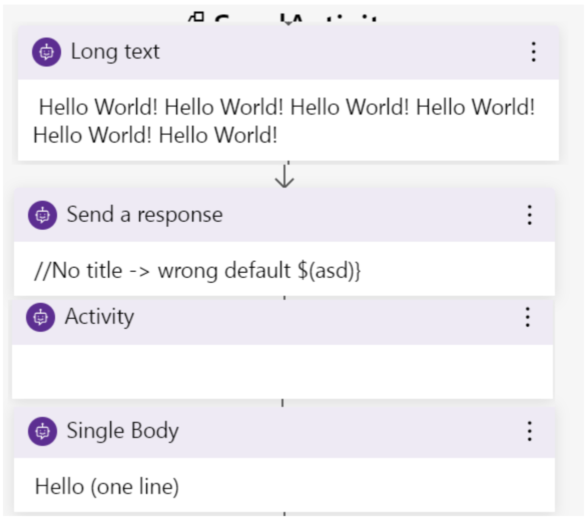
I would like to introduce why we need a layout cache mechanism in Visual Editor by several examples.

Problem: we need to know node sizes before rendering

To make the layout of visual editor accurate, parent node should know the exact size of a child node, thus it can draw itself properly with edges connected and blocks placed properly.

However, it's not possible to get a DOM element's sizes before it's really mounted.

If we are blind to the size of a node before rendering, all elements in the editor will be narrowed together:


Long text
Hello World! Hello World! Hello World! Hello World! Hello World! Hello World!
Send a response
//No title -> wrong default $(asd)}
Activity
Single Body
Hello (one line)

Solution: measure node sizes by analyzing its data

We use absolute sizes to define element. There is a `measureJsonBoundary` module for measuring sizes before rendering and a `elementSizes` definition to manage all size numbers such as 200px width, 48px height.

With this pre-render measurer, evert node should be rendered as we expected.

For example:

```
const sendActivitySize = new Size(200px, 48px);
const choiceInputSize = new size(200px, 48px + data.choices.length * 20px)
const ifConditionSIze = new Size(
    trueBranchWidth + falseBranchWidth,
    Math.max(trueBranchWidth, falseBranchWidth) + 100px
)
```

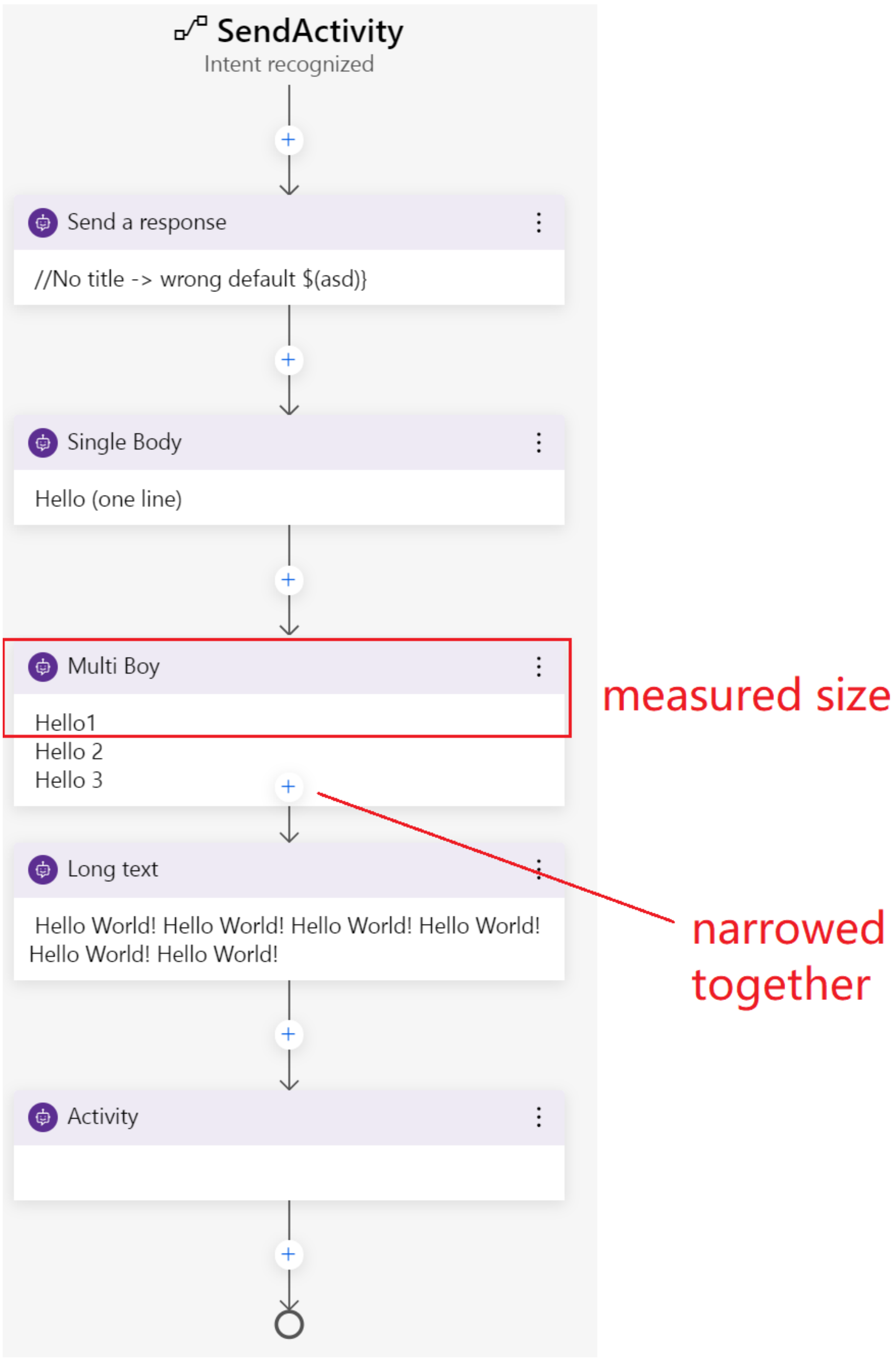
It's proved to be worked in our old version.

Problem: the size of a node is undetermined

After our new action design in which size of action content could be dynamic, we cannot measure node sizes correctly anymore.

Imaging we have a 'SendActivity' node, its content is 'Hello Hello Hello ..... Hello' which is super long. We couldn't determine how many lines we should spare to render it.

If we measure it a wrong size, editor will look like this:



Solution: measure DOM size after node mounted, bubbling up size changes and request a second rendering

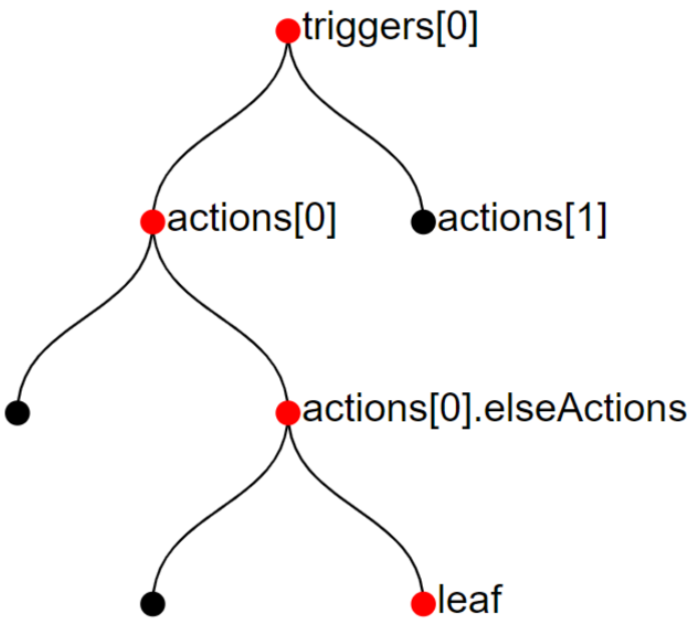
Use the `react-measure` lib to measure exact node size after first render cycle; then notify parents that the element's size is changed via `onResize` handler. It will trigger an extra rendering which causes flickering.

```
<Measure onResize={size => onResize(size)}>
    {({ ref }) => <Node ref={ref} />}
</Measure>
```

When a parent node receives new child size, it will recalculate its layout:

```
const ParentNode = ({ data }) => {
    const [layout, setLayout] = useState(calculateLayout(data));
    const updateLayout = (size, childId) => {
        const newLayout = calculateLayout(data, { childId: size });
        setLayout(newLayout);
    }
    return (
        <div width={layout.width} height={layout.height}>
            {...}
            <ChildNode id={childId} onReisze={size => updateLayout(size,
childId)} />
        </div>
    )
}
```

The whole editor's layout is in a tree structure



Size changes of leaf node will be bubbled up to all parents. So the layout of Visual Editor will always be updated.

Problem: adding / deleting nodes has flickering issue, switching between dialogs has flickering issue

Though the layout of visual editor will be always updated via a second rendering, from UX perspective users feel like the editor is flickering when adding / deleting / navigating between events.

This is because visual editor takes two render cycles to place elements at proper positions:

- in the first render cycle, editor mounts node blocks to estimated position with inaccurate estimated sizes;
- in the second render cycle, editor collects the measure results of DOM nodes and then adjust node blocks to their proper positions.

Flickering issue could be relieved by giving a more accurate estimation of node sizes; however, considering that we cannot estimate the accurate width of a string, we still need to do small adjustment after we get real sizes of DOM nodes.

So can we just cache calculated sizes and load them in the first render cycle? Then the 'estimated' size will always equal to the accurate size.

Solution: cache the calculated layout of unchanged nodes

Considering every node has a $designer filed which contains a unique id, we can optimize the flickering issue by caching sizes of those unchanged nodes.

The layout cache module is super straightforward:

```
class LayoutCache {
    private caches = {}
    public cacheNodeSize(id, size) {
        this.caches[id] = size;
    }
    public loadCache(id) {
        return this.caches[id];
    }
    public uncacheNodeSize(id) {
        delete this.caches[id];
    }
}
```

To consume the cache:

- If a node's size has been cached, before rendering, its size could be loaded from the cache directly instead of use the `measureJsonBoundary` which is inaccurate.

  ```
  const Node = ({ id, data }) => {
      const estimatedSize = loadCache(id) || measureJsonBoundary(data);
      return (
          <div width={estimatedSize.width} height={estimatedSize.height}>
              {something else}
          </div>
      );
  }
  ```

To maintain the cache:

- every time react-measure detected a DOM size change, the new size should be stored to the cache

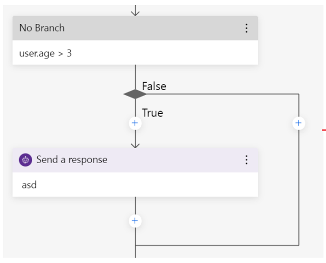
```
<Measure onResize={
  size => {
      // 1. notify its parent to bubble up resize event
      parent.onResize(size, id);
      // 2. maintain the layout cache
      LayoutCache.setSizeCache(id, size);
  }
 }>
   {({ref}) => <Node ref={ref} />}
</Measure>
```

- every time a node is removed, its cached size should be removed, along with all its parent nodes' cached sizes (because of as a tree, parent nodes' sizes are always determined by children)

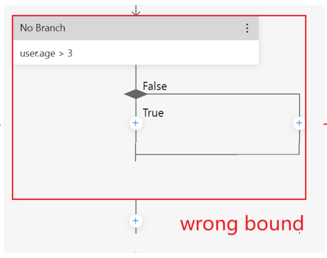
```
// class LayoutCache.uncacheNode
function uncacheNode(id) {
    const parentIds = findAllParents(id);
    [...parentIds, id].forEach(x => {
        delete this.caches[x];
    })
}

function deleteNode(id) {
    // 1. maintain layout cache
    uncacheNode(id);
    // 2. manipulate json
    const newData = jsonTracker.deleteNode(id);
    onChange(newData);
}
```

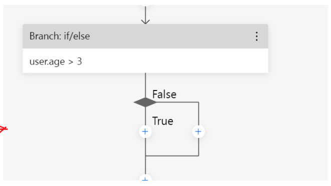
If we don't remove outdated sizes, the editor will run into an unhappy condition:



initial state (delete the SendActivity)



parent action loaded old cache



corrected after second rendering