React Game of Life Implementation

- Whole Grid re-rendering when state.board changed (only one element deep in multi-dimensional array)

https://stackoverflow.com/a/51136076/2176546

- Nested state not necessarily ideal
- React does shallow comparison for state & props

https://dmitripavlutin.com/use-react-memo-wisely/

- Using memoisation because cells are purely functional
- Significantly faster

https://reactjs.org/docs/react-component.html#shouldcomponentupdate

- Manually determine whether or not a row should update
  - https://developmentarc.gitbooks.io/react-indepth/content/life_cycle/update/using_should_component_update.html
  - Need to manually implement deep comparison because React only does shallow comparison
- Massively faster
- Should we also use this for Grid?
  - React doesn't re-render the rows
  - It just checks whether the rows needs to re-render
  - Nope, as React doesn't re-render the rows

Once done Game of Life logic

- Speedy and excellent conditional rendering

Implementation 1 // Next Iteration Workflow

1. Get current board
   1. Select all rows
   2. For each row, get the child nodes (cells)
   3. For each cell, determine whether the cell is 'alive" (i.e. has the `alive` class) and push this value onto an array which stores the row's cells' alive states
   4. This gives us a 2-dimensional array of the form [[...]]
2. Loop through every row and cell, and determine whether the cell should be alive on the next iteration
   1. We count the number of neighbours the cell has, making sure to consider boundary cases (a cell in a corner will only have three neighbours, for example)
   2. We look at the current alive status of the cell
   3. We apply the rules to determine whether the cell should be alive or dead on the new iteration

3. Render the updated board
    1. Get each row and cell DOM node
    2. For each cell, check whether the new status is alive or dead, and add or remove the `alive` class accordingly

Results
Automatically iterated 20 steps of a glider in Chrome

preact