U4872964V : `model` is just plain wrong naming, in my opinion
U4F64AKQV : I think it's like that to match up with MV*.
U6GFNSEPR : ah yeah, it's a bit clearer to name it `initialModel`
U6GB56346 : <@U3SJEDR96> hmm, do you mean `text` 's signature is *not* a free type variable?
U3SJEDR96 : No, it is. And that can be unified with `Never`. You could have `foo : Html Msg` and define `foo = text "hello"`, too. That would be completely valid. The type of the expression `text "hello"` is still `Html a`.
U3SJEDR96 : It's no more special than saying `foo : List String` with a definition of `foo = []`. That empty list doesn't really "hold" anything, so it could be a `List Never`, too, but since it has type `List a`, the compiler is fine with you saying it is a more specific thing than that, even though the implementation doesn't _need_ to be that constrainted
U3SJEDR96 : ```htmlNever : Html msg -> Html Never
htmlNever elem = elem
```

U3SJEDR96 : the same reasoning for why that doesn't compile is that you can't do this:
U3SJEDR96 : ```foo : List a -&gt; List Never
foo xs = xs
```

U6D3ERLA1 : This is confusing:
U6D3ERLA1 : ```Function `foldr` is expecting the 1st argument to be:

    Point -> Array.Array Row -> Array.Array Row

But it is:

    Point -> Grid
```

U6D3ERLA1 : But my Grid type is:```
type alias Grid =
    Array.Array Row
```

U3SJEDR96 : right. And the function you give to `Array.foldr` takes an entry of the array you provide as input, as well as the value to accumulate into. For example `List.foldr (\x sum -> sum + x) 0 (List.range 1 4)`
U3SJEDR96 : in your case, it seems like you may be providing a `Point -> Array Row` function, but that's not something `foldr` can work with
U6D3ERLA1 : Hmm... I'm trying to loop through an array of coordinates and return a new updated grid with each iteration...
U6D3ERLA1 : [{x=1,y=2}, {x=2,y=2} ...] <| forEach (\point -> getUpdatedGrid point)
U6D3ERLA1 : Maybe I just missed the accumulator..
U3SJEDR96 : hold up, I'm not sure I'm following. It feels like you want to transform each point in your grid, in which case you'd want a function `Point -> Point` and use `Array.map`
U6D3ERLA1 : ```placeBoat : Point -> Int -> Dir -> String -> Grid -> Grid
placeBoat p l d k grid =
    List.foldr
        (\pt outputGrid -> putGridVal pt k outputGrid)
        grid
        (getCoords p l d [])
```

U6D3ERLA1 : This seems to do it - I think I was missing the accumulator `outputGrid`
U3SJEDR96 : Alright, gotcha :slightly_smiling_face:
```