U6FFD2QG0 : yeah, basically
U3SJEDR96 : I would suggest taking the contents of your `TimeDone` branch, putting it into a separate function, and replace that with```
TimerDone ->
    timerDone model
Tick newTime ->
    if .activeTime (decTimer model) > 0 then (newTime, Cmd.none) else timerDone model
````

U0LPMPL2U : I'd suggest extracting the common logic to a helper function and calling that from both branches instead of trying to send a `Msg`
U6FFD2QG0 : fair enough. Is this just not a typical way that a Cmd would be used?
U0LPMPL2U : You almost never want to just send Msg to yourself
U0LPMPL2U : Msg is meant to represent events from the outside world
U3SJEDR96 : No, a `Cmd msg` represents something for the runtime to execute asynchronously, after which it can call your `update` with the resulting `Msg`
U6FFD2QG0 : ok, that's good to know
U6FFD2QG0 : thanks!
U3SJEDR96 : you don't really need the runtime in order to call a function, though, so using a function to abstract the behaviour of "what should happen when your timer is done" is definitely the recommended approach
U37HUSJ4R : If I wanted to make this more generic:```
updatePaused : Bool -> Call -> Call
updatePaused newValue ({ controls } as call) =
  { call
    | controls =
      Maybe.map
        (\controls -> { controls | paused = newValue })
        call.controls
  }
```

U37HUSJ4R : so for example `paused` could be any field
U3SJEDR96 : There is no generic record-updater syntax; sadly
U37HUSJ4R : :cry:
U37HUSJ4R : So if I have `hold`, `pause`, `hangup` for example I need 3 functions and 24 lines of code :disappointed:
U3SJEDR96 : Not entirely true; but there's another thing... Can a call be simultaneously on hold, paused and hung up?
U37HUSJ4R : this is more state of the buttons
U37HUSJ4R : so if I call is on hold then the state is
U37HUSJ4R : ```paused : True,
hangup : True,
paused : False
```

U37HUSJ4R : but yes it could be all three
U37HUSJ4R : multiple different valid states here
U3SJEDR96 : I'd try to think of a better way to model those, though. In the meantime, you can use something like ```
updateControls : (Controls -> Controls) -> Call -> Call
updateControls op call =
  { call | controls = Maybe.map op call.controls }
```

U37HUSJ4R : I'd LOVE a better way to model these
U37HUSJ4R : but really can't think of one :disappointed:
U3SJEDR96 : ```updatePaused  : Bool -> Call -> Call
updatePaused newValue call =
  updateControls (\controls -> { controls | paused = newValue }) call
```

U3SJEDR96 : you'd still end up with 15 lines of code, including annotations, but it would be pretty clear what they all did. And if you decided that `call.controls` should be a `Result` rather than a `Maybe`, that's only a single line to change
```

U3SJEDR96 : as for your control-states.. I think having a `Status = Ongoing | Paused | OnHold | Hangup` (or something similar, depending on requirements) would make sense, with functions `canPause : Status -> Bool` etc
U3SJEDR96 : Though it might make sense for a call to be both onhold and paused, in which case that would be a fifth case