Kamarás Tamás

Review by VeryCreatives

Norbi:
Egesz jo megoldas, AWS-es demo oldallal. JS oldalrol vannak problemas reszek pl: a state DOM fuggosege ami nem tul elonyos. A formai megjelenesen lehetne faragni, hogy tagoltabb legyen, illetve: tesztek. (az indentalasi hitvitaba inkabb nem megyek bele :))

Barna:
Thank you for your submission Tamas!

ES6 is used.
App is deployed to AWS beanstalk! (very cool).
Lightly tested.
(small typo in callToPlay() function return string)

Since the states have no particular order to them there are no benefits of mapping them to numbers. On large scale systems it would benefit performance especially when storing them in a database but here readability would be more important. (currently we have to keep in head which state maps to which number).

Snacks can be given to the child while he is hiding, this is incorrect.
A snack is given to the child when he is called to play or eat, this is incorrect.
Mood raise increment is incorrect.

I know separation was a concern but at the moment nor the stateMachine nor alterBySnack is really separated. stateMachine even depends on the DOM through changeBackground() function.

Ask these questions while inspecting the code to really see where separation would be needed:
Can any function in alterBySnack be tested separately without passing in a child and mutating it?
Can any function in stateMachine be tested separately without passing in a child and mutating it and without a document (not in browser)?

At the moment the answer is no for these questions. We also have to use a verbose API to mutate child's state. `app.stateMachine.callToKindergarten(pete)` this could have been `pete.callToKindergarten()`.

An example written in Elixir lang where state change and mood increment concerns can be tested separately:

```
def next_state(:call_to_play, :eating), do: :playing
def next_state(:call_to_play, :hiding), do: :playing
def next_state(:call_to_eat, :playing), do: :eating
def next_state(:call_to_kindergarten, _), do: :hiding
def next_state(_, _), do: :error

def mood_after_snacks(0), do: 0
def mood_after_snacks(n) do
  mood_after_snacks(n, 0)
end
def mood_after_snacks(0, mood), do: mood
def mood_after_snacks(n, mood) do
  mood_after_snacks(n - 1, mood + mood_increment(n))
end

def mood_increment(n) do
  cond do
    rem(n, 15) == 0 -> 8
    rem(n, 5) == 0 -> 4
    rem(n, 3) == 0 -> 2
    true -> 1
  end
end
```

I would advise researching “dependency injection” and solving the same exercise in an immutable functional language (for ex Elixir) which will force you to solve the exercise without mutations.

I really enjoyed this app and thumbs up for beanstalk deploy!