COMP 313/413
Group 6
Project 4

Report

Planning Phase:

In the initial phase of our project we worked on creating a state diagram for the Timer. We referenced the stopwatch state diagram presented to us in class and updated with the required states - Stopped, Increment, Running, Alarm. Then we proceeded to create an extended state diagram to further elaborate on the transition conditions between the states.

After establishing a state diagram we created a Github repository for the project and cloned the stopwatch app to use as our base code for the timer. At this time we started to think about the class relations and how we can go about refactoring the existing code.

Development Phase:

- We started off by identifying the new classes required based on the existing classes in the stopwatch app.
- Created new classes for Incrementing and Alarmed states while refactoring the code for the running and stopped states.
- We updated the DefaultStopWatchStateMachine class according to our new timer state diagram.
- We subsequently updated the related interfaces.
- Our four main states were connected through a state management class that was used to receive end user events and relay it to the required states.
- Each state upon receiving event signals would then determine the following state based on the predetermined conditions.
- We updated the UI to display two digits for the countdown.
- We then proceeded to add the sound notifications for the countdown beginning and the alarm.
- Throughout the development phase we used local branches on our computers and pushed our changes to the main repo in phases.
- Once our timer was running as expected we added inline comments for readability and future reference.

Testing Phase:

- We came up with the following list of Unit tests based on our state diagram:
    - Test initial state of the timer (timer stopped and time is zero
    - Test that the button increments the time when the timer is stopped
    - Test that the time does not exceed 99 and starts running after reaching max

    - Test that the timer starts running 3 seconds after the button is pressed
    - Test that the timer decrements time when running
    - Test that the timer resets to zero when the button is pressed while running
    - Test that the alarm starts beeping when time reaches zero without the button being pressed
    - Test that the timer continues in its current state when a rotation happens
- The unit tests were written based on each of the fundamental requirements.
- Through testing produced one bug with the alarm which was easy to identify and fix.

Relationship between extended state machine and code

Similarities:

- Both represent states, transitions, and events that trigger state changes.
- Both follow the same general logic

Differences:

- The extended state machine model does not highlight error handling.
- The ESM does not include low level details.
- The ESM does not take in consideration performance

It is definitely recommended to model first and then code as modeling the ESM give us a holistic idea of what the functional requirements are and how they can be implemented on a high level. It also allows us to establish the required conditions needed for transition between states. This makes it easier to not only approach the development in a modular way but also allows us to think about testing conditions from an early stage.

Upon implementing our code and carrying out the required test, our state diagram and our code are congruent with each other and needs no further updates.