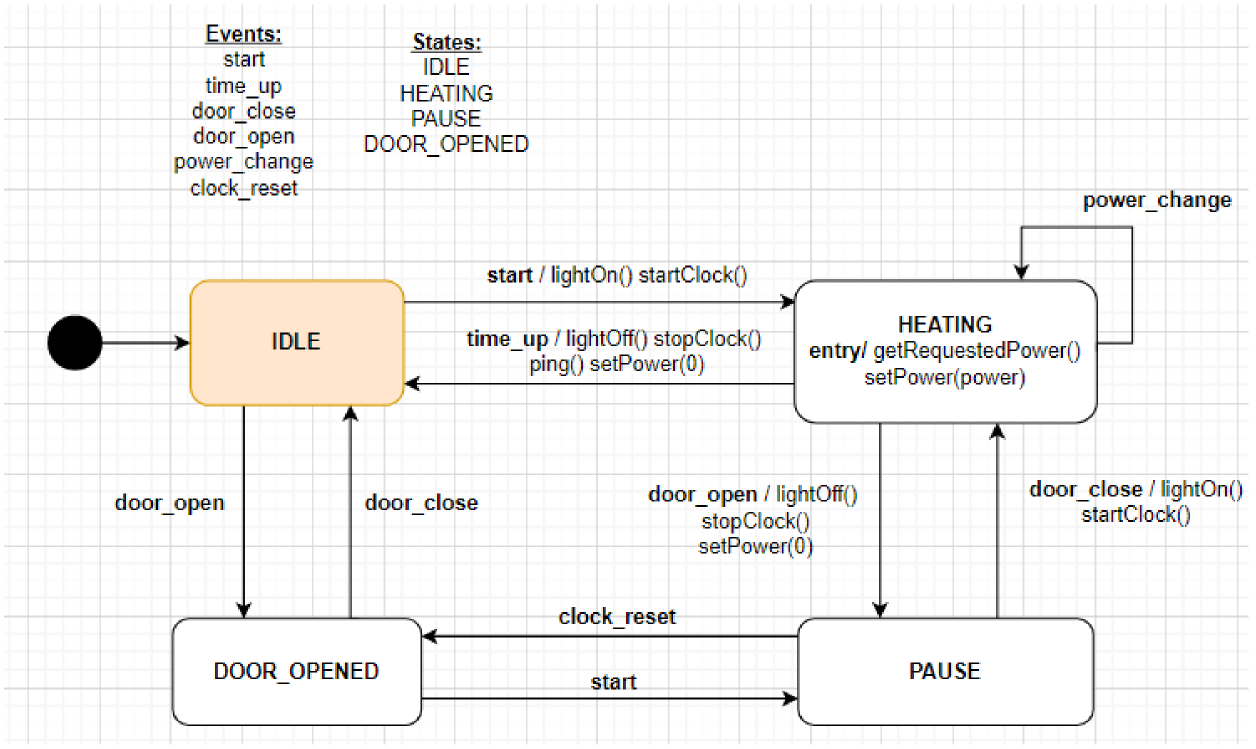
Microwave design project

State diagram of Microwave


Events:
start
time_up
door_close
door_open
power_change
clock_reset
States:
IDLE
HEATING
PAUSE
DOOR_OPENED
power_change
start / lightOn() startClock()
IDLE
time_up / lightOff() stopClock() ping() setPower(0)
HEATING entry/ getRequestedPower() setPower(power)
door_open
door_close
door_open / lightOff() stopClock() setPower(0)
door_close / lightOn() startClock()
clock_reset
start
DOOR_OPENED
PAUSE

In this state diagram you can see that we have 4 basic states which represent the whole system. Idle is for waiting for input from the user and representing an ideal state from where the heating process could start from. The start event contains the setting up of the microwaves timer and pressing the start button. Door opened state is self-explanatory, if the start event is triggered everything is ready, but the door is open so that has to be closed. Paused state represents a quite interesting state. Everything is set up, but the door is not closed so the heating cannot start. Heating state represents the main functionality of the microwave, the food is being heated. The state does not change in case of a different power setting while the heating state is active and running. If the heating time is up everything set back to default value and the microwave is back to the idle state.

Implementation of Microwave

It consisted of extending the states and the possible events as well as finishing the implementation of the microwaves code. I made one handle function for every state. In those functions all actions are put in a switch case which is separated for all relevant events, for events that are not relevant nothing happens. In every incoming event based on the actual state that the microwave is in a different kind of appropriate action is called. The actions are performed through the functions of the hardware layer interfaces which are stored in the microwave as attributes. Later by passing in actual objects, that implement these interfaces, the real implementation and hardware specific behavior of the microwave could be observed. The entry event of the heating state is implemented so that, every time a transition to that event is called, the specific entry action (get-setRequestedPower()) is called to make sure it is actually called in every time the state is entered.

Testing of Microwave

Testing is done by the gmock and gtest libraries from googletest. You can mock the interfaces that the microwave uses. All classes in the /test folder are mocks which implement the given interface it mocks. With that you pretend that the mocks are real working objects and instances so their function can be called. The microwave than receives these objects in its constructor thus uses these mimicked functionalities. This object is than set up as the main starting configuration of my tests, since this would have to be done each time. With EXPECT_EQ and the EXPECT_CALL built in testing functions, we can easily test the state changes for every event and all the function calls that happen along the way. With Times() WillOnce() WillRepeatedly() the amount times a specific function is called can be tested while keeping the return values right value as well.

Personal:

I tried to implement the test cases using the CMakeLists.txt files on windows, it did not work. I would appreciate some insight on why is that the case. Here are my 3 files:

Microwave/ CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.10)
project(Microwave)

set(CMAKE_CXX_STANDARD 14)

include_directories(product)

add_subdirectory(product)
add_subdirectory(test)
add_subdirectory(lib/googletest)
```

Microwave/product/CMakeLists.txt:

```
set(BINARY ${CMAKE_PROJECT_NAME})

file(GLOB_RECURSE SOURCES LIST_DIRECTORIES true *.h *.cpp)

set(SOURCES ${SOURCES})

add_executable(${BINARY}_run ${SOURCES})

add_library(${BINARY}_lib STATIC ${SOURCES})
```

Microwave/test/CMakeLists.txt:

```
set(BINARY ${CMAKE_PROJECT_NAME}_tst)

file(GLOB_RECURSE TEST_SOURCES LIST_DIRECTORIES false *.h *.cpp)

set(SOURCES ${TEST_SOURCES})

add_executable(${BINARY} ${TEST_SOURCES})

add_test(NAME ${BINARY} COMMAND ${BINARY})

target_link_libraries(${BINARY} PUBLIC ${CMAKE_PROJECT_NAME}_lib
gtest_main)
```